DEVELOPMENT OF SOPHISTICATED UNMANNED SOFTWARE SYSTEMS AND APPLICATIONS TO UAV FORMATION

DONG XIANGXU
(B.Eng, Xiamen University, China)

A THESIS SUBMITTED
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING
NATIONAL UNIVERSITY OF SINGAPORE

2012

Acknowledgments

First of all, I would like to express my deep and sincere gratitude to my supervisor, Professor Ben M. Chen for his guidance, encouragement, and patience during my Ph.D. studies at the National University of Singapore. His wide knowledge, systematic way of thinking have been of great value to me not only during my Ph.D. course but in my daily life as well.

I also wish to express my sincere thanks to Professor T. H. Lee, Professor Yew Kai Lum and Dr. Hai Lin for providing me suggestions and assistance during my academic studies.

Special thanks are given to our NUS UAV research group in the Department of Electrical and Computer Engineering, National University of Singapore. I will never forget the days when working with my team mates day and night. Particularly, I would like to thank Dr. Miaobo Dong for providing me detailed guidance in programming and debugging. Also, I really appreciate the technical suggestions from Professor Biao Wang, Dr. Guowei Cai, Dr. Feng Lin, and Mr. Fei Wang. I am also grateful for the generous help from Dr. Kemao Peng, Professor Delin Luo, Mr. Beiqing Yang, Mr. Jinqiang Cui, Mr. Kevin Ang, Mr. Swee King Phang, Mr. Shiyu Zhao, Mr. Ali Karimoddini, Miss Jing Lin and Mr. Kun Li.

Moreover, I am very grateful to my flat mates during the last years in Singapore. They have been providing me encouragement, support and joy in my life all the time and make me feel at home. I wish to thank Miss Xiaolian Zheng, Dr. Sen Yan, Dr. Lingling Cao, Mr. Bo Tian, Mr. Xiangjing Zhang, and Mr. Xuetao Chen.

Last but certainly not the least, I owe a great debt to my parents for their everlasting love, care and understanding during my whole life.

Contents

Summary

Unmanned aerial vehicles (UAVs), have achieved great improvement and widely deployed in both military and civilian applications during the last two decades. With the improvement of sensing technology and powerful processing processor, more advanced intelligent tasks can be achieved with UAVs. The software system plays a vital role in the UAV systems to realize effective resource allocation, task scheduling, automatic control and wireless communications. A systematic and universal software design methodology will facilitate the UAV deployment in various applications, which is the focus of this thesis.

This thesis first presents a framework for multiple unmanned systems which contains onboard systems and one ground control system (GCS). The framework is designed to be universal that can be applied to unmanned systems such as unmanned aerial vehicles (UAV), unmanned ground vehicles (UGV), autonomous underwater vehicle (AUV) and etc. The framework can also incorporate more unmanned systems with inter-vehicle communications to realize team coordination and control. A software architecture is also provided to better understand the role of each module in the framework.

Based on this framework, each software component is analyzed and designed. The UAV onboard system needs to perform tasks including flight task scheduling, hardware data retrieval, device operations, control law calculation, communications and servo driving. A multi-thread architecture is carefully designed to ensure real-time performance of onboard system. A task scheduling method is implemented to coordinate the execution of all tasks within the assigned time slot. Various software modeling diagrams such as data flow dia-

gram, Unified Modeling Language (UML) based diagram are deployed to model the onboard system with a top-down perspective. A behavior-based control law implementation is proposed to realize complex automatic flight tasks.

Another important part of the onboard system, the onboard avionic system design and operating system (OS) customization, are also explored. The image customization is the fundamental layer on which the onboard software is built. A complete onboard avionic system consisting of flight control subsystem and vision subsystem are presented for two kinds of processors. For control subsystem, an industry-standard real-time operating system (RTOS), QNX Neutrino is customized with the Board Supporting Package (BSP) provided by QNX. For vision subsystem, the open source Linux is adopted and customized to support vision applications.

To realize flexible communications within UAV systems, a hybrid communication architecture is established. With inter-UAV communications, more advanced tasks such as formation flight can be achieved with information exchange among UAV team members. A novel communication approach based on the telecommunications is also implemented.

The last part of the framework is the GCS which performs the tasks including sending user commands, receiving flight status data, displaying the data in various perspectives. The document-view class based on Microsoft Foundation Class (MFC) is adopted as the software architecture for GCS. The GCS can perform hardware-in-the-loop simulation with real-time data update from onboard systems.

A behavior-based scheduled flight and a fast forward flight are conducted to verify the successful design of the software system. A multiple UAV formation application with a leader and a follower is also accomplished thanks to the multiple UAV support of the software system. Finally, this universal software system are ported to another different UAV platform within a short time and high level control performance is also achieved.

To conclude, the contributions of this work are summarized and promising future directions are provided.

List of Tables

List of Figures

Nomenclature

Latin variables

$\mathbf{a}_{nr}$	acceleration reference in NED frame
$\mathbf{A}$	state matrix of the linearized model
$\mathbf{B}$	input matrix of the linearized model
c	Euler angle in NED frame
F	state feedback matrix
G	feed forward matrix
h	NED frame altitude
p	body frame rolling angular velocity
$\mathbf{p}_{nr}$	position reference in NED frame
q	body frame pitching angular velocity
r	body frame yawing angular velocity
$\mathrm{T_s}$	the sampling period of the onboard software
u	body frame x axis velocity
v	body frame y-axis velocity
$\mathbf{v}_{nr}$	velocity reference in NED frame
w	body frame z axis velocity
y	body frame y-axis position
z	body frame z-axis position

Greek variables

θ	pitching angle in NED frame
ϕ	rolling angle in NED frame
ψ	yawing angle in NED frame

Acronyms

A/D	Analog-to-Digital
ARM	Advanced RISC Machine

AUV	Autonomous Underwater Vehicle
BSP	Board Supporting Package
CF	Compact Flash
RPT	Robust Perfect Tracking
CPU	Central Processing Unit
DARPA	Defense Advanced Research Projects Agency
D/A	Digital-to-Analog
DC	Direct Current
DFD	Data Flow Diagram
DSP	Digital Signal Processor
GCS	Ground Control System
GPS	Global Positioning System
GUI	Graphical User Interface
HILS	Hardware-In-the-Loop Simulation
INS	Inertial Navigation System
IMU	Inertial Measurement System
I/O	Input / Output
IP	Internet Protocol
KB	Kilo Bytes
MAV	Manned Aerial Vehicle
NED	North-East-Down
NUS	National University of Singapore
OO	Object Oriented
OpenCV	Open source Computer Vision
P2P	Peer to Peer
PC	Personal Computer
PCB	Printed Circuit Board

RC	Radio-Controlled
ROI	Region Of Interest
RPM	Rotations Per Minute
RS232	Recommended Standard 232
RTOS	Real-Time Operating System
SBC	Single Board Computer
SD	Secure Digital
STD	State Transition Diagram
TCP	Transmission Control Protocol
UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle
UART	Universal asynchronous receiver/transmitter
UDP	User Datagram Protocol
UML	Universal Modeling Language
USB	Universal Serial Bus
WLAN	Wireless Local Area Network
WiFi	Wireless Fidelity
2D	Two-dimensional
3D	Three-dimensional
3G	Third Generation

Chapter 1

Introduction

1.1 Overview

Unmanned Aerial Vehicles (UAVs) are becoming an increasingly important area in both military and civilian applications. With the capabilites such as aerial photography, remote reconnaissance, cargo carrying, team cooperation among multiple UAVs etc., more research focus have been put on the UAV systems. Specifically, small-scale UAVs have great interest to research people due to its small size, low cost, and outstanding maneuverability. With the rapid development of microprocessors and sensors, a hobby-based flight vehicle can be upgraded to a high-end UAV with advanced intelligent capabilities. With this mobile yet flexible platform, various research activities can be implemented and verified. This motivates our NUS UAV research team to construct various kinds of UAVs including single-rotor helicopter, coaxial helicopter and quad-rotor helicopter to fulfill different application requirements such as GPS-based navigation, vision-based target detection and tracking, formation flight etc.

Developing a UAV platform is quite challenging due to the following reasons: 1) Mechanical design of avionic system should consider weight balance and maintenance. 2) The onboard avionic system hardware components should be systematically selected to satisfy

different applications. 3) The UAV itself is a highly dynamic plant, thus an accurate model should be obtained and a sophisticated control law must be properly designed. 4) Software system design requires coordinating the hardware components and computing automatic flight control algorithms. 5) The practical flight tests also require a pre-designed procedure to accomplish the flight mission safely.

In what follows of this chapter, necessary technical background is first presented in Section 1.2 to better understand the UAV topic. In Section 1.3, the motivations and contributions of this thesis are introduced. Finally, the outline of this thesis is given in Section 1.4 for convenient reference.

1.2 Technical Background

In this section, the technical background covers three topics. First, the working principle of UAV software system is introduced. Then the software design methodology of real-time UAV system including onboard system and ground control system is presented. Finally, the various applications of UAV systems are covered.

1.2.1 Working Principle of a UAV System

An UAV system, is an aircraft without a human pilot on board. Its flight is either controlled autonomously by computers in the vehicle, or under the remote control of a navigator, or pilot on the ground or in another vehicle [71]. As explained, the ultimate goal is to achieve automatic flight control without human interventions. To realize such a system, a bare UAV frame is first purchased from a commercial hobby shop. Then various hardware components including power system, sensors, processors, actuators, etc. are integrated and mounted on the UAV. This small computer system is called onboard avionic system. With onboard system, the automatic control capability can be realized via retrieving sensor data, calculating control signals and outputting to actuators to control the movement of UAV. The UAV airframe can be a fixed-wing or a single-rotor helicopter. The flight dynamics

of various airframes will determine the suitable application for each kind of UAV. For example, a fixed-wing UAV can perform patrol mission along the coast due to its fast speed, power saving design. On the other hand, although single-rotor helicopter possesses highly dynamics and is difficult to control compared to fixed-wing UAVs, its capability of hovering makes it advantageous in the tasks such as target detection on the ground. To obtain the necessary navigation information, the conventional inertial measurement unit (IMU) plus GPS is deployed in the avionic system. The onboard computer software system performs complicated algorithms with data from IMU and GPS to calculate the driving signals.

With the onboard autopilot system constructed, a ground pilot is usually needed to act as a navigator to control the UAV by a radio-controlled (RC) transmitter. A ground control system (GCS) is applied to monitor the flight data of UAV. Meanwhile, it can upload user commands to onboard system to realize remote control.

1.2.2 Real-time Software Design

As explained in Section 1.2.1, the UAV software system plays a significant role to realize the automatic control purpose. It consists of two main parts: one is the onboard system and the other is the GCS. As UAV performs critical flight missions, a real-time performance must be achieved for onboard avionics system. The embedded real-time software design is more difficult than it is for the traditional desktop PC. The real-time system design must consider timeliness, robustness and safety which non-real-time system does not concern. The real-time software system will interact with various peripheral hardware sensors operating at different input/output (I/O) speeds. Before calculating onboard control laws, all the sensors data should be retrieved and processed. Thus the time consumed by each peripheral sensor should be considered to ensure the data used for calculating control laws is most recently retrieved. The real-time software system is also designed in a robust approach that given a certain hardware failure, the control task can continue executing without halting and thus provides graceful performance downgrade. As UAV flight task is inherently risky

to human beings, a safety precaution method also needs to be addressed in software design.

Another challenge is that the development tools for real-time systems are usually hosted on PCs and workstations but targeted to smaller, less-powerful and less-resources onboard processors. In this case, a cross-compiler must be generated on development hosts to deploy the executable files on target machines. Such development environment makes the debugging of target machines extremely difficult because of the lack of sophisticated online debugging tool chains and proper configurations.

Given such a challenging specifications in real-time system, an efficient real-time software design methodology should be proposed to facilitate the whole development process. Various real-time design approaches can be classified into two categories: structure-oriented design and object-oriented (OO) design. In [21], the structure-oriented design can decompose the system specifications in a top-down approach with the aid of data flow diagram (DFD). After exploring the details of the data transforms in DFD, a task identification can be carried out based on the criteria such as time-critical functions, functional cohesion, I/O dependency and etc. The object-oriented design based on universal modeling language (UML) in [17] focuses on the software modeling of complex systems including real-time requirements. An object can represent things that contain both data and behaviors. The real objects such as aircrafts, sensors, and engines can all be described in the object perspective. Both methods are capable of describing the real-time system in a static and dynamic perspectives. However, the object-oriented design also addresses the software issues such as maintenance and scalability.

The overall real-time systems consist not only the above mentioned onboard applications, but the under layer operating system as well. A real-time operating system (RTOS) determines the real-time performance fundamentally. Thus the operating system customization also needs exploration. With the customized OS images, we can realize high-level hardware interactions, efficient task scheduling and other valuable services provided by the RTOS.

In [38], a cooperative software consisting of both avionics and ground station is effi-

ciently developed. One highlight is the 802.11b communication mechanism adopted which provides both long distance with relative large transmission power and flexible peer to peer network which greatly facilitates the cooperative information exchange among UAV team. Also, the RTOS is selected as QNX 6.2 which is an industrial standard operating system suitable for critical real-time tasks. A team control interface is developed in ground station to make the team coordination task easier for the ground operator. The successful demonstration of cooperative surveillance with three UAVs verifies the overall software system design. In [29], the Stanford DragonFly UAV is also designed based on the real-time QNX. The software architecture possesses modular and flexible features which are critical for experimenting with different configurations. Currently, there are also quite a lot of open source sofware available such as Paparazzi [56], OpenPilot [55], ArduPlane [47]. The open source community provides not only the software but the detailed hardware design as well. However, all these projects are for general purpose UAV applications and thus not optimized for research-oriented computation intensive tasks.

To realize user interactions with onboard system, a GCS software should be designed in a user-friendly way, which GCS operator can clearly monitor the flight status and upload user commands easily. Besides, some mission planning tasks such as waypoint uploading can also be conducted on GCS. To realize graphical user interface (GUI), there are a lot of available tool-kits, such as Labview, visual basic, microsoft foundation class (MFC) and so on. All development kits provide mechanisms to facilitate GUI layout with user interactions. For research purpose, a small and portable laptop based GCS is prefered. The QGROUNDCONTROL station is an open source project developed at ETH [61]. The QGround GCS consists of three perspectives: engineering view is designed for professionals to explore the flight status and behaviors, the user mode view is for amateurs to interact with UAVs, the pilot view is specifically for ground pilot to perform remote control with joystick signals. Another widely adopted GCS is the ArduPilot Mission Planner [48] from DIYDrones. It has the function to upload the correct onboard software system to different platforms.

1.2.3 Formation Flight

As the demand of employing UAVs in practical cooperative flight tests has been increasing due to the new operation paradigm [3, 27], many research groups are contributing to this area. In the perspective of formation flight, several practical flight test platforms are proposed. In [25], an efficient platform consisting of eight fixed wing UAV are implemented. The computer cluster is adopted as the ground station for high-burden coordination algorithm execution. A high speed Ethernet network is deployed to dispatch the reference signals to the eight UAVs. Furthermore, a practical cooperative formation task with two UAVs is performed to verify the overall platform. But the centralized coordination architecture will not facilitate the robustness of a distributed system. Since all critical coordination commands come from the ground station, this architecture will render the whole UAV team a failure if the ground station cannot perform its task in a proper manner. At Stanford STARMAC [22], a simple but efficient multiple-UAV is adopted. The UAV is quad-rotor based, which is robust, easy to construct compared to the traditional complex helicopters and also superior to the fixed-wing UAV in the maneuver flexibility. Also, the UAVs operate in a distributed approach where the optimization is performed on the UAVs simultaneously. However, the communication link is selected as Bluetooth which limits its long distance operation capability. Also, as the ground station is developed with Labview, the GUI development period can be shortened, but it will definitely be lack of real time performance and flexibility of interacting with different communication hardwares. In [1], the cooperative platform is also based on fixed-wing UAVs with a high level coordination approach, i.e., hybrid control system. With the hybrid methodology, the coordination task can be formulated within the hybrid framework and represented with automaton formula, thus the commands for each UAV can be derived automatically and efficiently. The supervisor under the hybrid framework is able to capture both physical dynamics and switching logic. However, the supervisor implemented in a centralized approach also renders the potential single node failure problem.

1.2.4 NUS UAV Research Applications

NUS UAV research team has been developing UAVs since year 2003. During the last few years, we have successfully developed various platforms of small-scale UAVs. The first generation of UAV is the HeLion [8], a Raptor 90 helicopter famous for great maneuverability. With accurate modeling and control law, HeLion has achieved very good control performance [6]. With the obtained experiences, a twin helicopter, SheLion, is constructed. Compared with HeLion, SheLion has incorporated the vision subsystem to accomplish vision related missions with the collaboration of flight control subsystem. With the automatic capabilities, these two UAV helicopter are assigned as a leader and a follower in the formation flight.

Meanwhile, our research focus has moved to even smaller UAV platforms to realize indoor navigation applications. A coaxial helicopter, FeiLion, is developed for this purpose. The FeiLion as shown in Fig. 1.2 also possesses automatic flight capability and vision processing power. The previous flight control software system on HeLion and SheLion has successfully been ported to FeiLion onboard processors. The processor architecture on FeiLion is based on ARM while the processors on HeLion and SheLion are based on X86.

In May 2012, our team decided to participate the UAVForge competition organized by DARPA at USA. The competition requires a backpack sized UAV to accomplish a series of challenging tasks. A fully customized coaxial helicopter GremLion as shown in Fig. 1.3 is constructed with more advanced capabilities such as semi-auto control with the aid of ground pilot, long range video transmission, vision-based feature detection and tracking. Thanks to the flexible and cross-platform software design, all the UAV helicopters can adopt one single copy of onboard applications which greatly shortens the development and deployment period on new UAV platforms. As such, the development of GremLion can be finished within five months to be ready for competition.

Figure 1.1: UAV helicopter - HeLion

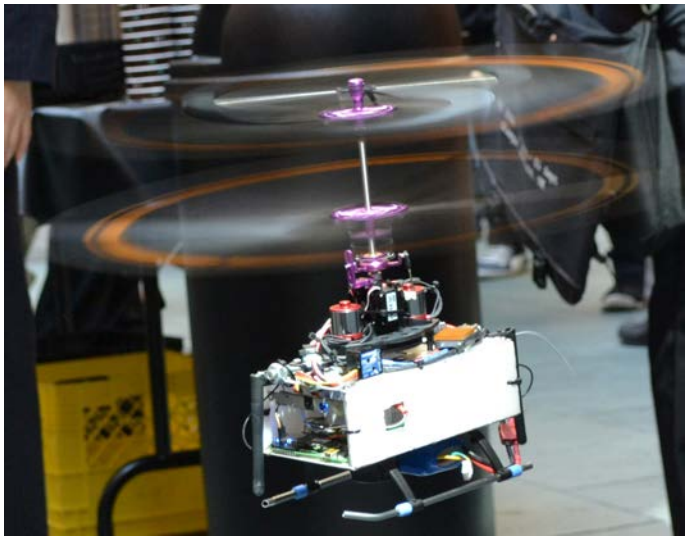


Figure 1.2: UAV helicopter - FeiLion

Figure 1.3: UAV helicopter - GremLion

1.3 Motivation and Contributions of This Research

This thesis aims to develop a universal software framework not only suitable to UAVs but also applicable to other unmanned systems. With this software systems, various kinds of applications such as waypoint-based navigation, vision-based target tracking, cooperative tasks such as formation flight can be conducted and verified on different UAV platforms. In addition, although onboard software system has been implemented and introduced in a number of applications, there are little research on a systematic approach to designing a real-time software system with the combination of DFD and UML modeling tools.

1. In this thesis, we have proposed a highly effective, scalable and robust architecture for control and coordination of multiple UAVs. A hybrid communication architecture is implemented to realize efficient information exchange among UAVs and GCS.

2. A scalable, efficient real-time software system for the UAV systems including onboard avionic system and ground control system are developed. The onboard software system is designed with an object-oriented approach based on UML diagrams with the aid of DFD. The sophisticated onboard development environment including target operating system image customization and hardware interface intergration are also introduced.

3. A number of practical flight applications are conducted on different UAV platforms to verify the realiable and efficient design of the software systems.

1.4 Outline of Thesis

The outline of this thesis is organized as follows: first a universal framework of multiple UAVs is introduced and analyzed. Based on the proposed framework, the real-time software design of onboard modules are presented in Chapter 3. The software platforms of onboard avionic system including real-time operating system image customization and generation are presented in Chapter 4. In Chapter 5, the software implementation of GCS is provided. In Chapter 6, the communications among UAV and GCS are introduced, where the communications architecture, communications protocol are developed. Finally, various practical applications are conducted to verify the successful design of the software systems in Chapter 7.

Chapter 2

Framework of UAV Systems

2.1 Introduction

As introduced in Section 1.2.1, a UAV is an aircraft that is controlled by a computer, or under the control of a navigator, or pilot on the ground. Given this description, a complete practical UAV system can be represented in Fig. 2.1. Based on this figure, the specifications of a UAV system are listed as follows:

1. An airframe equipped with a receiver can perform stable flight by a radio-controller from a ground pilot.

2. Various sensors to retrieve navigation data, such as position, velocity, acceleration, Euler angles, angular rate and so on.

3. Onboard computer to receive sensor data and calculate control algorithms based on these sensor data.

4. Servos to receive control signals from computer and drive the movements of the airframe.

5. The computer can simulate the dynamics of the airframe with a simulation model.

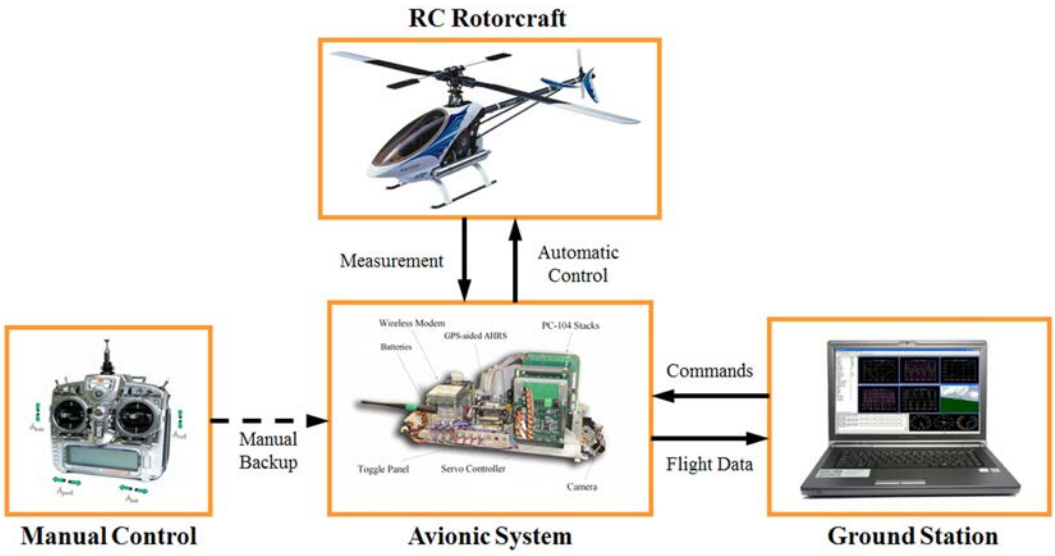


Figure 2.1: A complete practical UAV system [4]

6. Can send flight data to ground pilot and can receive commands from ground pilot.

7. A ground operator to operate a ground control system. The ground control system can receive flight data from UAV and can upload user commands to UAV.

2.2 Framework

Onboard System

Given the specifications in Section 2.1, and take into account the scenario with more UAVs, the framework of multiple UAV systems can be illustrated in Fig. 2.2. It consists of two main components: onboard system and ground control system. Each UAV onboard system has six main modules: *Unmanned vehicle*, *Simulation model*, *Sensing* (sensor data acquisition and processing), *Flight control* (automatic navigation and control), *Servo driving* (control movement of actuators) and *Wireless communications* (vehicle-vehicle and vehicle-ground communications). The sensor data acquisition, navigation and control and UAV dynamics construct the control loop. Specifically, a UAV model is built into the onboard software

to realize hardware-in-the-loop simulation. Besides, the vehicle-vehicle communication is applied for cooperative data exchange to feed to cooperative control module to realize UAV team cooperative control, such as formation flight control. In addition, the flight status data of each UAV is transmitted back to ground station via the vehicle-ground communication. While the user can send commands to each UAV with this communication link.

Ground Control System

For the ground control system, it consists of three modules, *Wireless communications* between GCS and UAVs, *Information monitoring*, and *Task management*. The *Information monitoring* can be realized in several approaches, such as the dynamic graph view, Google Maps view, and 3D rendering. A *Task management* module is also deployed to assign individual tasks such as hover, cruise flight to individual UAVs. In addition, the *Task management* can assign a team task to a group of UAVs to accomplish a collaboration goal. Meanwhile, the receiving function in the communication module feeds the UAV status data to the *Information monitoring* for various ways of rendering.

With this framework, all necessary UAV modules including onboard system and ground control system functions are clearly structured and presented. The logical data flow among different modules also facilitates the design and analysis of UAV systems. On the other hand, the framework incorporates both hardware and software modules. For example, the modules such as *Sensing*, *Servo driving* and *Wireless communications*, involve the interactions with the corresponding hardware. Finally, this framework can be further decomposed where each module can be designed systematically in a top-down approach and maintain the correct data flows in the whole UAV systems.

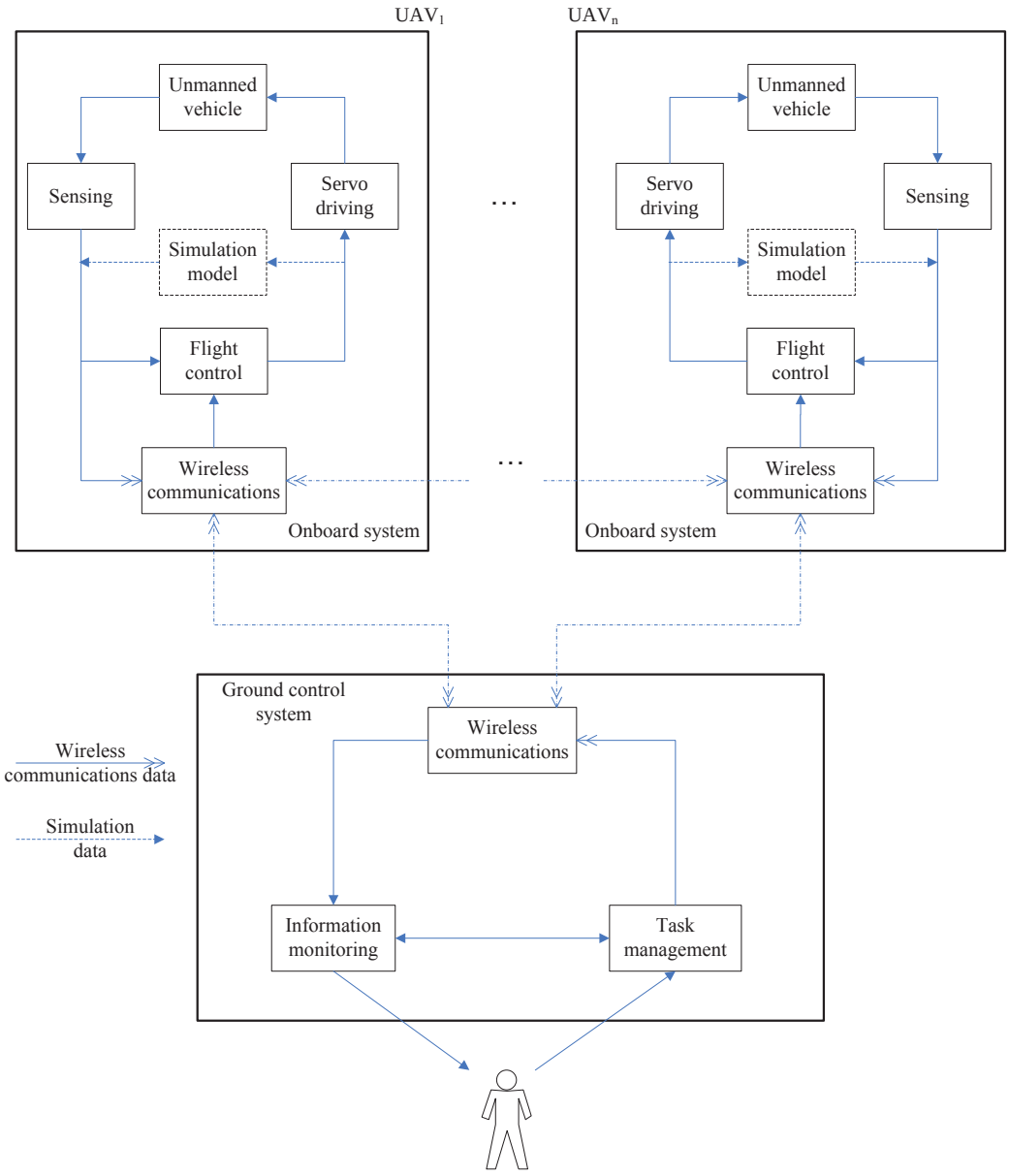


Figure 2.2: Framework of UAV systems

2.3 Software Architecture

The above framework incorporates both hardware, software and control algorithms. To better understand the role of software systems within the framework, a software architecture with different layers are presented in Fig. 2.3. Basically, both UAV and GCS can be considered as a computer system consisting of peripheral hardware, operating system and user software. Specifically, for UAV systems, the hardware include the aforementioned sensors, actuators, communication modules, A/D device, processor and etc. The operating system reside in the processor which is responsible for interfacing with peripheral hardware. The user software consists of user data, user programs and protocols. It is in the top layer of the software architecture. The flight control related programs are referred at this layer. As shown in the figure, there are various layers in the software system starting from the bottom bootloader to the top control related applications. Each lower layer provides services requested by the upper layer, such as hardware drivers, file operations, networking services and so on. All these layers are executed in the avionic processor. The interactions between processor and peripheral hardware are achieved via the rich hardware interfaces provided for the processor. The software system design thus involves all the layers from hardware interfaces to user applications. The details of each part are provided in the following chapters.

2.4 Conclusion

This chapter proposes a flexible and universal framework for unmanned systems including various platforms of unmanned vehicles and a ground control system. With this framework, it can facilitate the analysis and design of various aspects of the whole systems such as navigation and control for single UAV, cooperative control and coordination among multiple UAVs and etc. In addition, the implementation details such as hardware integration and software design can also be represented with a top-down approach under this framework.

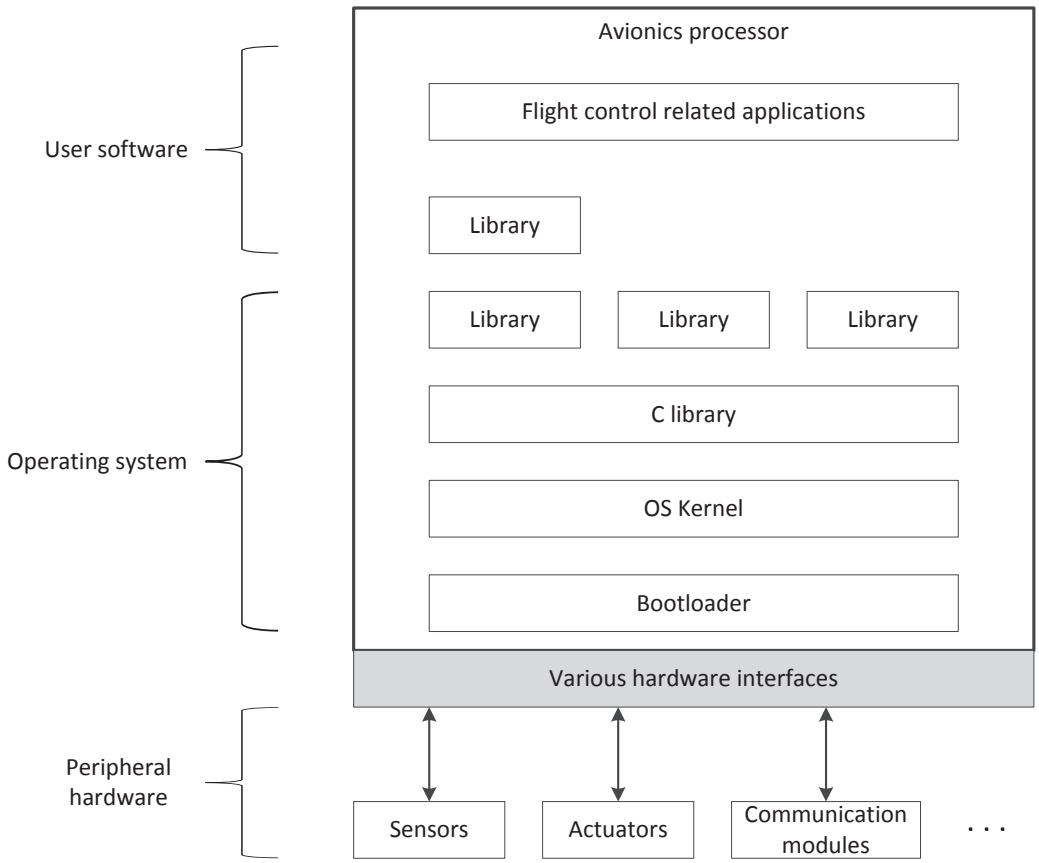


Figure 2.3: Software architecture

Chapter 3

Onboard Systems

3.1 Introduction

The onboard system is the most critical component in the UAV systems which must satisfy real-time performance. It consists of two main layers: the top layer is the user application to realize automatic control or vision related functions. The under layer is the operating system which provides hardware driver support, task scheduling, network service and etc. to the user application in the top layer. The architecture of the embedded avionics system is illustrated in Fig. 3.1. As shown in the figure, the avionic applications such as flight control and vision processing lay in the top layer of the whole embedded system. The layer below consists of the common libraries developed by the user or third party. The C library is the interface between user applications and kernel. Subsequently, the OS kernel contains the services for upper layers such as memory management, network stack and device drivers. The bootloader at the bottom will perform hardware initialization during power-up, load and execute the kernel services. Therefore, the onboard avionic system development involves two tasks: one is the operating system selection and customization based on peripheral hardware, the other is the onboard user application. The under layer development also needs to consider the integration of peripheral sensors given the hardware interfaces. As

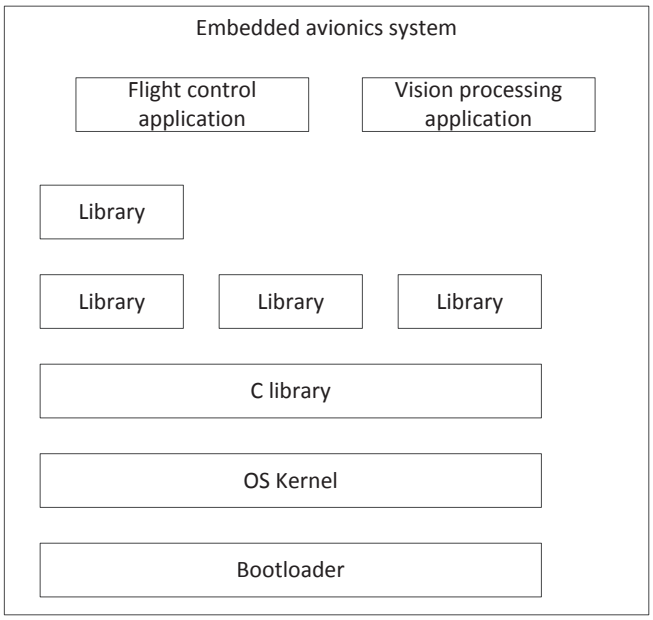


Figure 3.1: Software architecture of embedded avionic system

such, we combine all the development of under layer works as software platform for the onboard avionic system.

For top layer application development, as introduced in Section 1.2.2, there are structure-oriented and object-oriented design approaches. Both methods have their own modeling diagrams for analysis and design. For structure-oriented design, a data flow oriented method is used to decompose the system specifications into detailed data transforms which are represented with bubbles. Because real-time systems are usually data flow oriented, where data in the systems can be considered from input to output, the DFD method can be adopted in system analysis. The DFDs are developed and decomposed to sufficient depths until major subsystems or subtasks are identified. Next, the object-oriented design with UML is used to represent the properties and behaviors of each subtask. With UML class diagram, the structure property of the onboard system can be obtained. With state transition diagram (STD), the system dynamic behaviors can be achieved. The onboard subtasks will also need

to collaborate with one another to finish a common goal, then the message sequence diagram can be drawn to represent the interactions among different objects. With the combination of DFD diagrams and UML diagrams, the onboard user application can be correctly modeled and thus implemented.

We will address the development of top layer user applications in this chapter while the software platform is discussed in the next chapter for a better organization of the thesis.

3.2 Unmanned Vehicles

The *Unmanned vehicle* refers to a UAV airframe body as illustrated in Fig. 2.1. The UAV plant body is a practical UAV airframe platform such as a traditional single rotor helicopter, co-axial helicopter, quad-rotor helicopter or even ground vehicles. Based on application requirements, the proper UAV model should be selected and mounted with avionic system to realize automatic flight. For indoor application, the small size UAV should be selected such as FeiLion. For outdoor navigation purpose, HeLion and SheLion equipped with GPS can be deployed. If long endurance is required, then a fixed-wing UAV should be adopted correspondingly. As the onboard avionic software is designed to be universal, the vehicle platforms can be deployed as many as possible.

3.3 Sensing

3.3.1 Logical Representation

The detailed logical representation of sensing block is shown in Fig. 3.2. As shown in the figure, the sensing data come from different sensors, it can be IMU, GPS, ultrasonic, laser scanner or vision subsystem, though the hardware details are not directly listed at this level. All the sensing data are fed into the data selection and processing unit, where sensor data will be selected and combined.

A more general and complicated scenario is described in Fig. 3.3, where all sensors

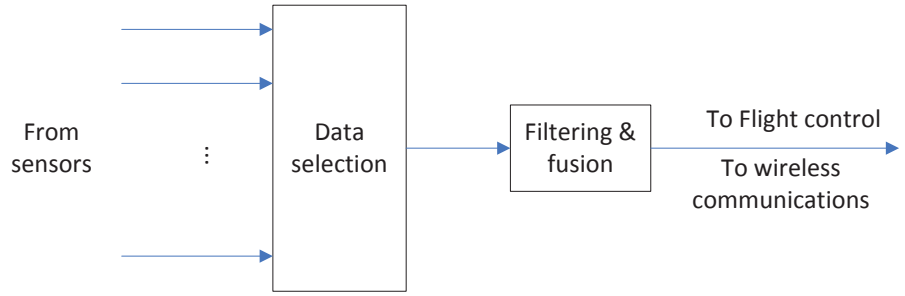


Figure 3.2: Sensing description in level 1

including IMU plus GPS, laser scanner, ultrasonic and vision subsytem are incorporated. The sensors selected finally are decided based on the UAV platform. For example, the *Sensing* on HeLion is shown in Fig. 3.4.

The sensors deployed in the HeLion avionic system consist of IMU plus GPS and ultrasonic. The navigation data output from IMU plus GPS is first filtered to smooth the position and velocity output, then fused with the height near ground from ultrasonic sensor. Specifically, the fusion is conducted in such a way that the height output in the navigation data is used for flight control system most of the time except when the UAV is performing near ground operations such as take-off and landing.

3.3.2 Inertial Measurement Unit

Among the sensors, the Attitude Reference Heading System (AHRS) plus GPS are the core of onboard system. IMU plus GPS provide the most fundamental sensing data for the aircrafts. The critical data provided are 3-axis accelerometer in NED frame, 3-axis gyro in body frame, ground velocity in NED frame, and GPS LLA (Latitude, longitude, altitude). Only with the correct updated data from IMU module, reliable automatic flight control can be realized. On GremLion platform, a good performance IG500N from SBG system is selected as the AHRS module.

Based on the datasheet of the sensors, the software modeling of the sensing block is

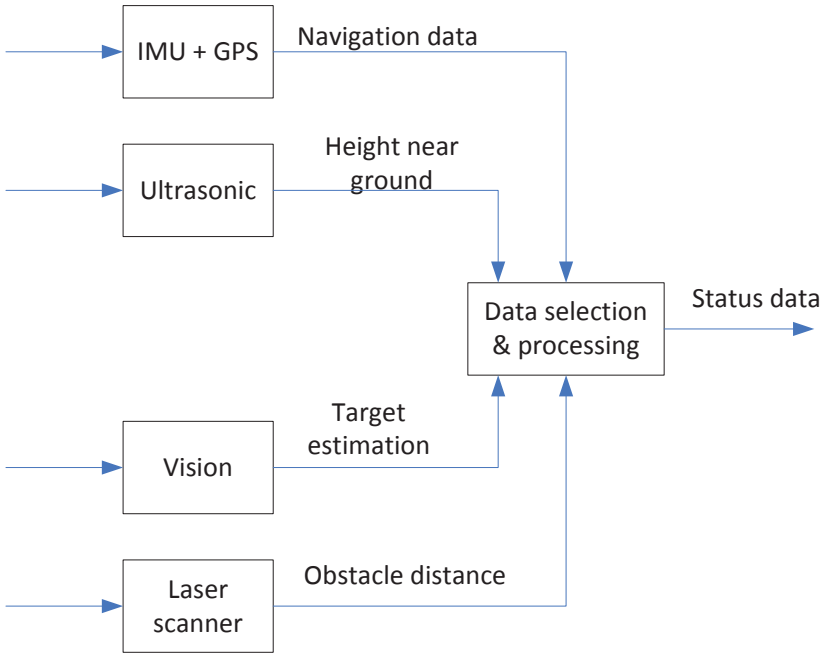


Figure 3.3: Sensing description in level 2

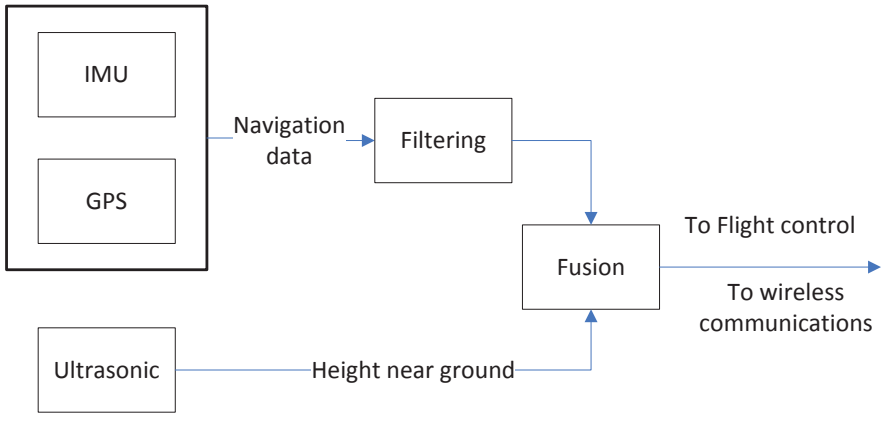


Figure 3.4: Sensing description in level 2 - HeLion

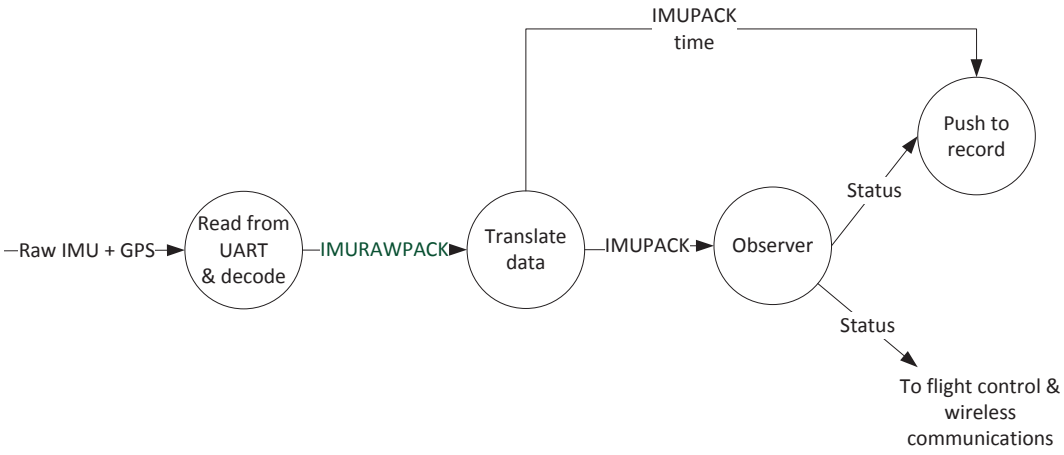


Figure 3.5: Level 3 data flow diagram of Sensing

illustrated in Fig. 3.5. The real-time data flow is first extracted from the UART buffer with serial port read operation. After decoding the data in binary format, the output data is in *IMURAWPACK* format. After the translation transformation, the data become user readable format *IMUPACK*. Since the sensor raw data output contains high frequency noise, a filter transformation is applied to the *IMUPACK* packet. For most modern IMU with GPS combination, the measurement can provide position, velocity, angular rate, Euler angles and angular rate. However, the mathematical representation of the helicopter model involves other unmeasureable variables where an observer is needed to estimate. On the other hand, when simulation mode is applied, the built-in helicopter model generates the same output as the IMU with GPS unit as a parallel data flow route. The sensor data processing and fusion are performed in another class called *clsState*. After estimation is done, the UAV status data is pushed to data store, flight control and communications modules. The data structure such as *UAVSTATE*, member variables and operations can be found in the class diagram in Fig. 3.6.

For a specific IMU device, some configurations are needed to define the output data, such as Euler angles, velocity, ect. and byte sequence as well. Fig. 3.7 shows clearly the

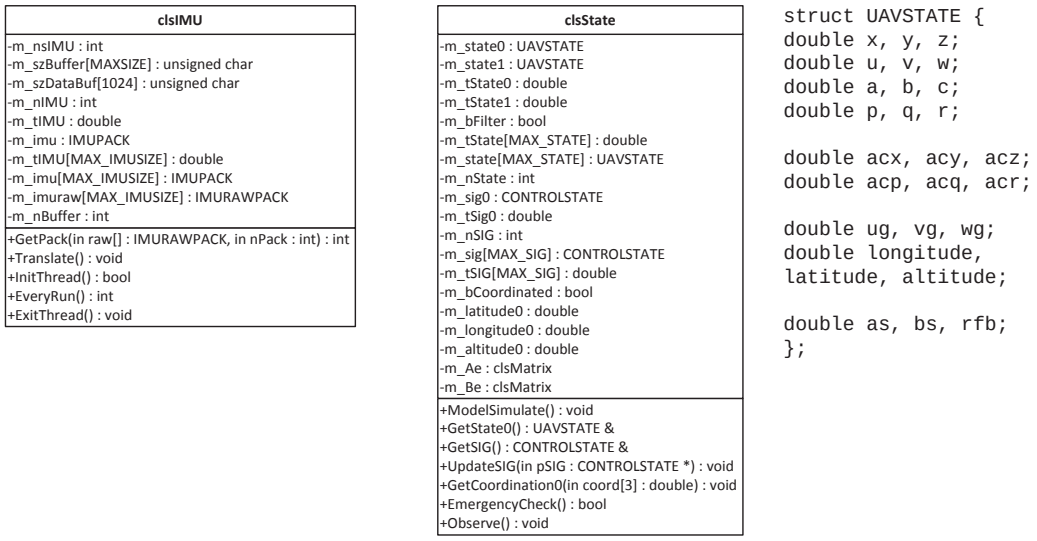


Figure 3.6: Class diagram - clsIMU

configuration of output sequence of IG500N.

After configuration, the onboard software will accomplish receiving and parsing of the incoming data. Below is the code snippet of receiving and parsing:

```
// receiving
int nRead = read(m_nsIMU, m_szBuffer, MAX_IM8BUFFER);
...
// euler angles
m_gp8.gp8.a = (double)(*(float *)(m_szDataBuf);
m_gp8.gp8.b = (double)(*(float *)(m_szDataBuf+4));
m_gp8.gp8.c = (double)(*(float *)(m_szDataBuf+8));
// body angular rate
m_gp8.gp8.p = (double)(*(float *)(m_szDataBuf+12));
m_gp8.gp8.q = (double)(*(float *)(m_szDataBuf+16));
m_gp8.gp8.r = (double)(*(float *)(m_szDataBuf+20));
// NED accelerations
m_gp8.gp8.acx = (double)(*(float *)(m_szDataBuf+24));
m_gp8.gp8.acy = (double)(*(float *)(m_szDataBuf+28));
m_gp8.gp8.acz = (double)(*(float *)(m_szDataBuf+32));
// GPS LLA
m_gp8.gp8.latitude = GETDOUBLE(m_szDataBuf+36+6) * PI/180;
```

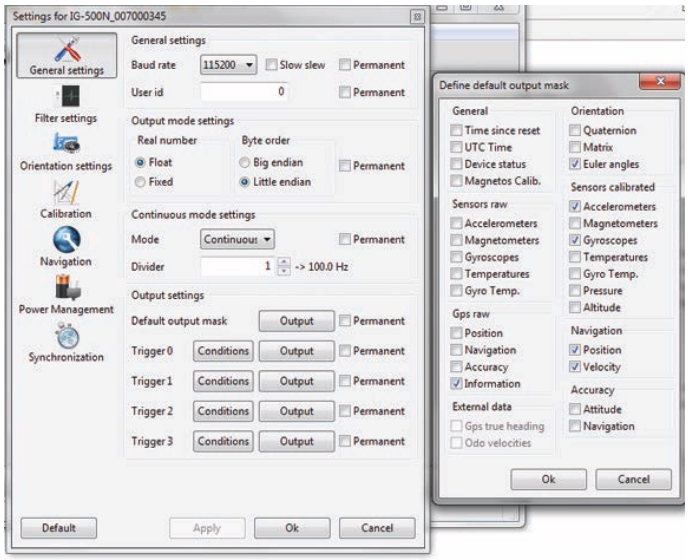


Figure 3.7: Configuration of IG500N

```
m_gp8.gp8.longitude = GETDOUBLE(m_szDataBuf+44+6) * PI/180;
m_gp8.gp8.altitude = GETDOUBLE(m_szDataBuf+52+6);
// NED velocity
m_gp8.gp8.u = (double)(GETFLOAT(m_szDataBuf+60+6));
m_gp8.gp8.v = (double)(GETFLOAT(m_szDataBuf+64+6));
m_gp8.gp8.w = (double)(GETFLOAT(m_szDataBuf+68+6));
```

3.4 Simulation Model

The simulation model block is to realize hardware-in-the-loop simulation. The simulation model can be represented as a black box as shown in Fig. 3.8. The black box has two sources of inputs and helicopter state output. In mathematical form, the HeLion UAV model [7] can be formulated as a 15 order ordinary differential equation (ODE) as below:

$$\dot{x} = f(t, u, v, x) \tag{3.1}$$

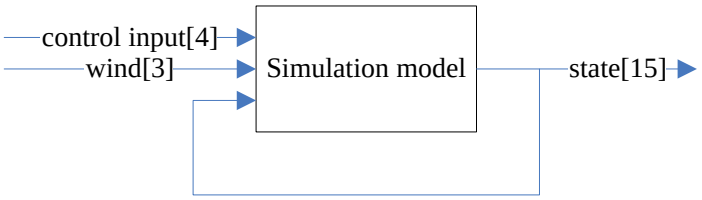


Figure 3.8: Simulation model diagram.

where x represents the UAV model output with 15 states, t is the current system time, u is the current control signal input with four input channels, and v is the wind disturbance in three directions.

For the ODE implementation, the classical Runge-Kutta approximation method is applied in the software. Given the initial UAV state and control input, the approximation can be conducted in the following approach:

$$\begin{cases} \dot{x_{k_1}} &= f(t_k, u_k, v_k, x_k) \\ \dot{x_{k_2}} &= f(t_k, u_k, v_k, x_{k_2}) \\ \dot{x_{k_3}} &= f(t_k, u_k, v_k, x_{k_3}) \\ \dot{x_{k_4}} &= f(t_k, u_k, v_k, x_{k_4}) \\ x_{k+1} &= x_k + 1/6T(x_{k_1} + 2x_{k_2} + 2x_{k_3} + x_{k_4}) \end{cases} \tag{3.2}$$

where T is the onboard main loop period, x_{k_1}, x_{k_2}, x_{k_3} represent the 1st, 2nd and 3rd state iteration, u_k is the control signal input, v_k is the wind disturbance, x_k is the current state variable, and x_{k+1} is the next state variable derived.

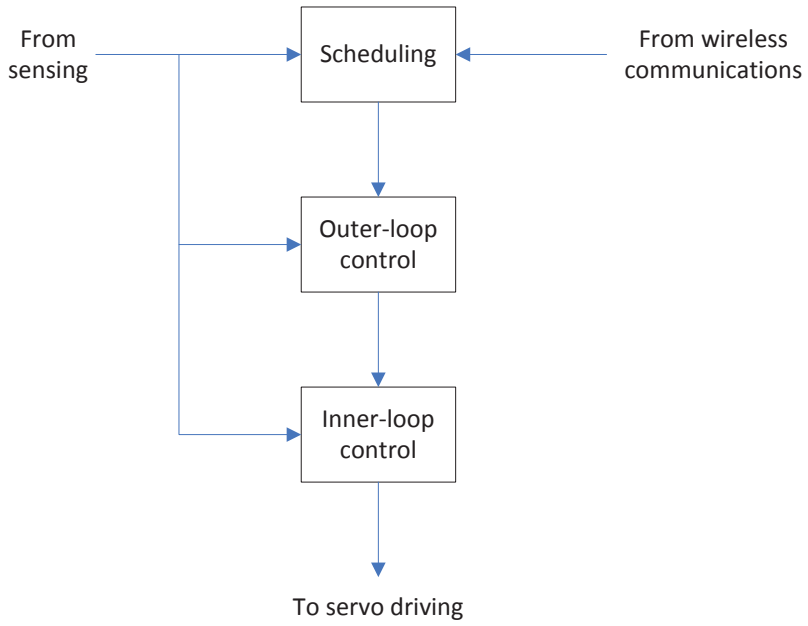


Figure 3.9: Flight control description in level 1

3.5 Flight Control System

3.5.1 Logical Representation

The level 1 description of flight control is shown in Fig. 3.9. The flight control module consists of mainly three units: task scheduling, outer-loop control and inner-loop control. The task scheduling is to generate the outer loop references given current status data and user commands from wireless communications. Based on the reference signals, the outer-loop realizes position, velocity and heading control by generating references for the inner-loop. The inner-loop is to stabilize the UAV attitude. During the control procedure, the output of inner-loop controller is sent to the servo driving module to drive the actuators on the UAV.

3.5.2 Software Modeling

With the top-down conquered logical representation of the onboard system, the software modeling approach is thus necessary to model the static functionalities and dynamic behaviors within the logical representation. The data flow diagram is given in Fig. 3.10. In this diagram, there are two parallel data processing at the initial stage. One route of processing is the control commands in the data store of user control commands, which is processed via the *Process commands* and further stored in the data store of *Behaviors and paras*. Another route is the emergency checking process, which determines the state is in emergency or not which is explained in later section. Then both data processing routes are fed into *Set control flags*, where a decision table is used to select the control blocks based on the *Behaviors and paras* data store and *Emergency status*. Given different flags, the corresponding outer-loop and inner-loop controllers are activated as listed in Table 3.3.

Process Commands

In the *Process commands* bubble, the user input commands from telemetry data are further parsed into UAV behaviors and corresponding parameters for further control algorithm usage. A further detailed level of *Process commands* is shown in Fig. 3.11. Based on different control code, the corresponding behavior and its parameters are extracted out in this process. For example, when the user issues the "hold" command from GCS, the current UAV transfers to hover behavior with current position and heading. Similarly, when a path tracking command is received, the UAV will start tracking the path with the assigned number of pre-defined path. Considering all the behaviors the UAV will perform, a series of commands is listed in Table 3.1.

UAV Behaviors and Parameters

Table 3.2 lists the parameters given the corresponding behaviors. When the UAV is in the behaviors such as hold, emergency, engineup and enginedown, no specific parameters are

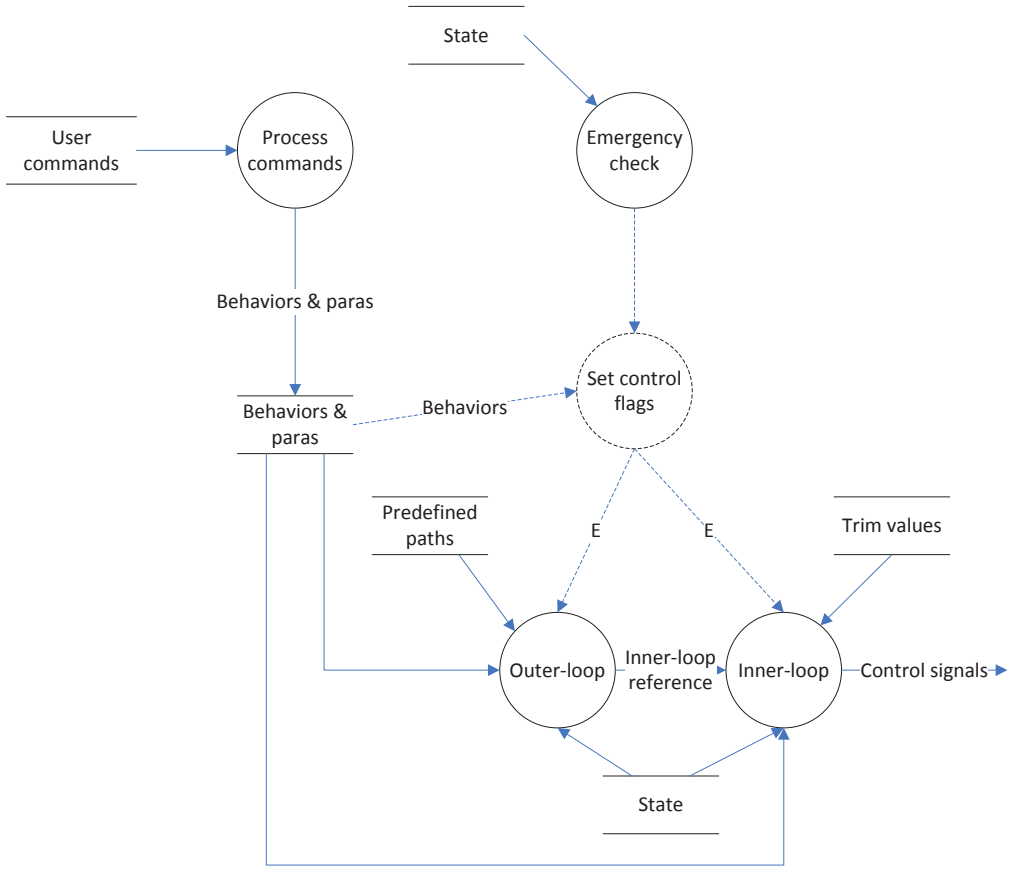


Figure 3.10: Level 3 data flow diagram - Flight control system

Table 3.1: Command lists

Command	Description
run	Start the onboard tasks, should be executed at first
quit	Exit the onboard tasks when the flight task is finished
engine(level)	Hold the engine at a specification level. The commands can be engine(low), engine(medium), engine(high), or engine(number) where number specify the level of engine scaling from 0 to 1. While no parameter is specified, engine then default represent the command engine(low). The engine/engine(low) command is mostly first used when the test begin from the ground.
hold	Hover at the current position. The command is mostly used before the helicopter perform specific tasks in the air. And the pilot switch to autonomous after this command was sent.
hfly(x,y,z,2dvel)	Fly to position (x,y,z) with respect to current position with a 2D velocity 2dvel
path(n)	Perform a path tracking task given the path number n, the paths can be generated offline and online.
plan(n)	Perform a scheduled flight task, currently plan(1) is full envelop flight including take off and landing
para(n)	Set the parameters for onboard control block, n is the predefined parameter identity.
test(n)	Perform newly developed test for onboard system, can be integrated into new command system once testified.

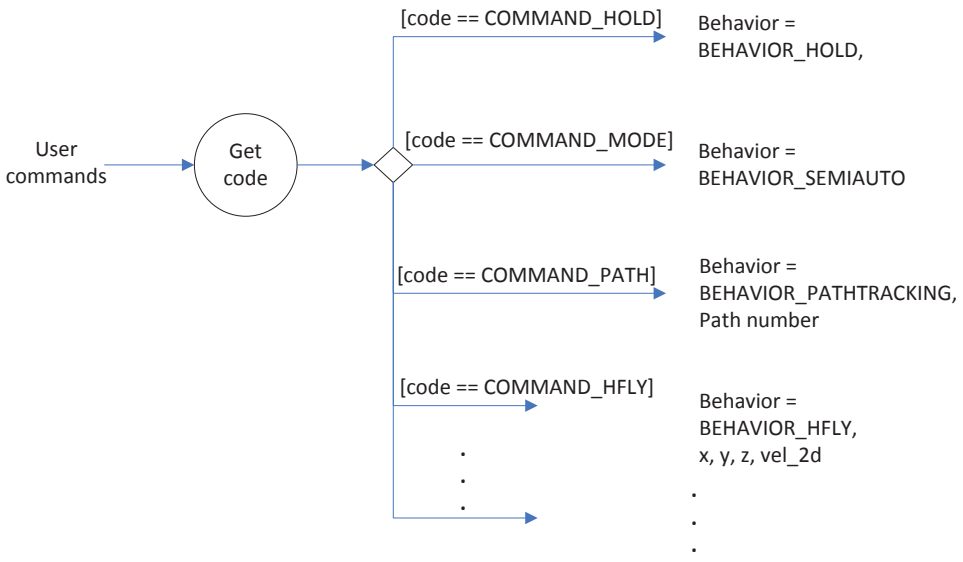


Figure 3.11: Data flow diagram - Process Commands

needed. Instead, different control blocks are activated as described in the above pseudo code. For hold behavior, the control blocks needed are outer-loop controller and inner-loop controller. For emergency, only a specially designed inner-loop is needed, where only the trim values of the servos are directly outputted. In engineup and enginedown, it is the rotations per minute(RPM) of main blades being controlled during take-off and landing. From the perspective of implementation, most of the tasks can be categorized into path tracking behavior where take-off path and landing path can also be integrated. The emergency, engineup and enginedown are three special behaviors which should be designed separately from the path tracking. Please note that each UAV platform has the emergency behavior, while the engineup and enginedown behaviors are only suitable for the Raptor 90 helicopters HeLion and SheLion. In the case of path behavior, the parameter of the path number is provided, and the position reference as well as velocity reference are calculated at every control loop and fed into outer-loop control law as references $[\mathbf{p}_{nr}, \mathbf{v}_{nr}, \mathbf{a}_{nr}]$.

Table 3.2: Control behaviors

Behavior	Parameters	Description
hold	null	keep current position and heading
fly	u, v, w, r	fly with a specified velocity
hfly	x, y, z, 2dvel	fly in 2D with a velocity of 2dvel
path	path number	track path(n)
emergency	equ	when status is abnormal
engineup	throttle low to high	when engine starts running
enginedown	throttle high to low	when engine stops running

3.5.3 Control Law Implementation

The control law implementation follows the data flows shown in Fig. 3.10. The inner-loop is to realize stable attitude control while the outer-loop is designed to realize navigation control given the position and heading reference generated from the scheduling layer. Specifically, an H_∞ control law [9] is applied to inner-loop and a RPT (Robust Perfect Tracking) control law [10] is applied to the outer-loop. The outer-loop is to generate references for the inner-loop as the input. The flight scheduling module is to divide the whole flight tasks into small behaviors such as take-off, path tracking, and landing. The flight tasks can be as simple as conducting an automatic hover operation, or can be complicated as a search and rescue command for a UAV team. With this hierarchical approach, various control schemes as well as high level tasks can be clear cut in logical representation and practical implementations. Specifically, for different UAV platforms, the corresponding blocks of outer-loop and inner-loop are activated with different control parameters. We note that although different control laws are executed given different UAV platforms, the control structure of both inner-loop and outer-loop are universal.

Flight Scheduling

The *Scheduling* resides as the top layer in the control 3-layer structure as shown in Fig. 3.9, which performs behavior related processing and then outputs outer-loop references for the outer-loop control law. Given a mission, it is fulfilled in the behavior-based approach [15]. This mechanism is more suitable for UAV task scheduling than the architecture proposed in [19, 33, 40]. The literature references focus more on the logical representation of the goals, actions, events, environments and etc. which needs much more computation power to derive the actions automatically. Given the low level tasks, it is reasonable and convenient to provide only the outer-loop references given different tasks. As such, considering the real-time computation requirements, the latter method is adopted within this thesis. The UAV flight behaviors can be identified as the following components: hover, head turn, longitudinal (forward/backward) flight, lateral (left/right) flight, vertical flight and customized paths. By concatenating various flight behaviors, a sophisticated flight task can be accomplished with one behavior after another. All the flight behaviors are further unified as a path tracking behavior. Given different behaviors, the path tracking can be realized by providing different outer-loop references. By providing the position reference and heading reference given at each time instance, the combination of different behaviors can be done in a timely fashion.

For the behavior-based control mechanism, task scheduling, behavior execution and control law calculation in each loop, the following pseudo code can be used to describe:

stage1: look up scheduled plan

check all events linked to current node

if any event occurs return corresponding behavior and parameters

else return null

stage2: translate current behavior into activation control blocks given current

Table 3.3: Control flags

UAV platform	Behavior	Control flags
GremLion	hold	B1, A1
	path	B1, A1
	hfly	B1, A1
	emergency	A2
HeLion, SheLion	hold	B5, A5
	path	B5, A5
	hfly	B5, A5
	emergency	A6
	engineup	A7
	enginedown	A7

UAV platform

if behavior equals hover, set control flags

else if behavior equals emergency , set corresponding parameters and control flags

else if behavior equals path, set corresponding path parameters and control flags

...

else do nothing; //behavior is null, no modifications on parameters, keep current control

stage3: execute control laws

run outer-loop function

run inner-loop function

The control flags with the corresponding control modules are summarized in the Table 3.3.

Example - GremLion

Take the platform GremLion for example. The GremLion is a complex co-axial helicopter with the control structure designed as shown in Fig. 3.12. The control structure is composed of the following critical blocks: *Task scheduling* which can switch between *Auto reference generation* and *Manual reference generation*, hardware platform *Unmanned aircraft*, sensing *IMU plus GPS*, *Outer-loop control law* and *Inner-loop control law*.

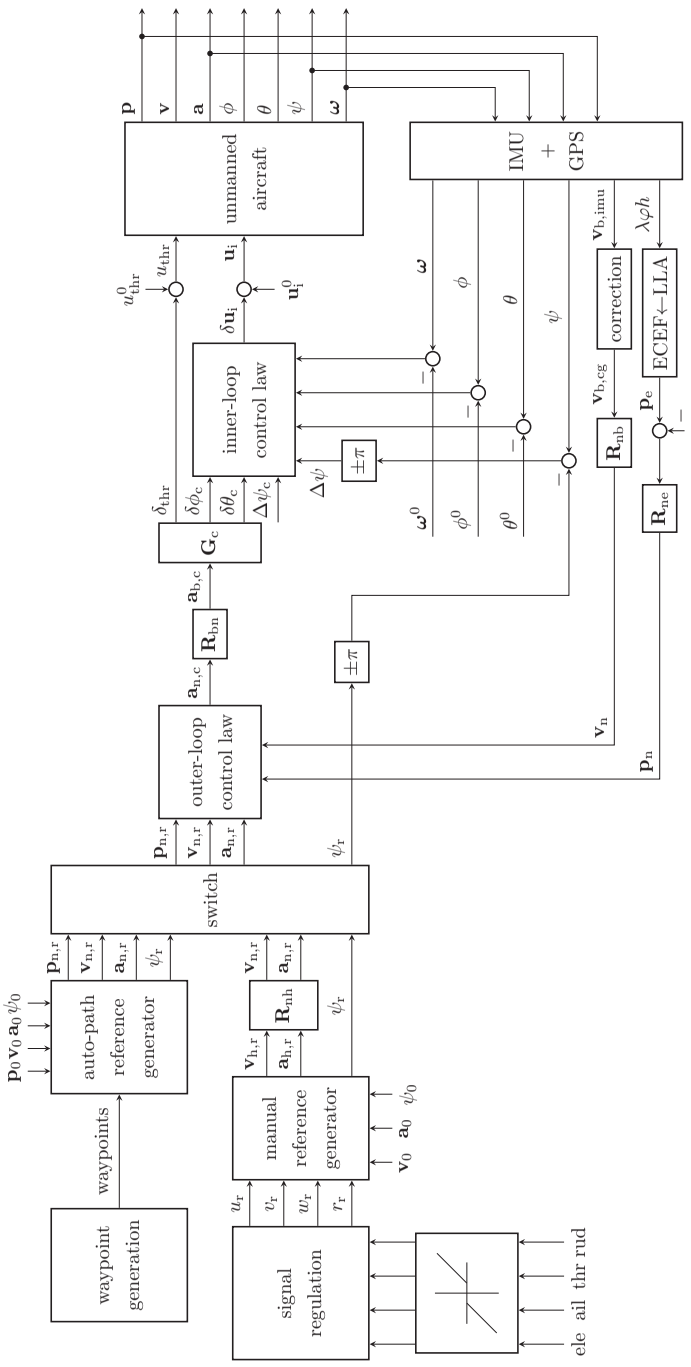


Figure 3.12: GremLion complete control structure

As the module *Flight control system* is the most critical part of onboard system, a more detailed description of the *Outer-loop control law* and *Inner-loop control law* is provided in Fig. 3.13. The three main modules in this figure are implemented via the following mathematical equations:

- Outer-loop control law

$$\mathbf{a}_{nc} = \mathbf{F}_o \mathbf{x_o} + \mathbf{G}_0 \mathbf{v_o}$$

$$\mathbf{x_o} = (\mathbf{p}_n, \mathbf{v}_n)'$$

$$\mathbf{v_o} = (\mathbf{p}_{nr}, \mathbf{v}_{nr}, \mathbf{a}_{nr})'$$

  where $\mathbf{x_o}$ composes the current status of NED position and NED velocity, $\mathbf{v_o}$ is the current reference for position, velocity and acceleration, all in NED frame. $\mathbf{F}_o$ is the state feedback gain, while $\mathbf{G}_o$ is the forward feedback gain designed via the RPT approach.

- Inner-loop control law

$$\delta_{\mathbf{u}_i} = \mathbf{F}_i \hat{\mathbf{x}}_i + \mathbf{G}_i \mathbf{v}_i$$

  where $\hat{\mathbf{x}}_i$ is the observer output, and inner-loop reference $\mathbf{v}_i = (\phi_c, \theta_c, \Delta\psi_c)'$. The matrices $\mathbf{G}_i$ and $\mathbf{F}_i$ are the control gains derived based on H_∞ approach.

- Observer

$$\hat{\mathbf{x}}_i = \int \mathbf{A}_i \hat{\mathbf{x}}_i + \mathbf{B}_i \mathbf{x_i} + \mathbf{K}_i \mathbf{v}_i$$

$$\hat{\mathbf{x}}_i = (\phi, \theta, \psi, p, q, r)' - (\phi_0, \theta_0, \psi_0, p_0, q_0, r_0)'$$

  where ϕ_0, θ_0, p_0, q_0, r_0 are the equilibrium values of GremLion during the hover flight condition, ψ_0 is the heading angle at the time when the inner-loop control law is activated. $\mathbf{v}_i$ is the inner-loop command reference, which is given by $\mathbf{v}_i = \mathbf{a}_{bc}\mathbf{G}_c$.

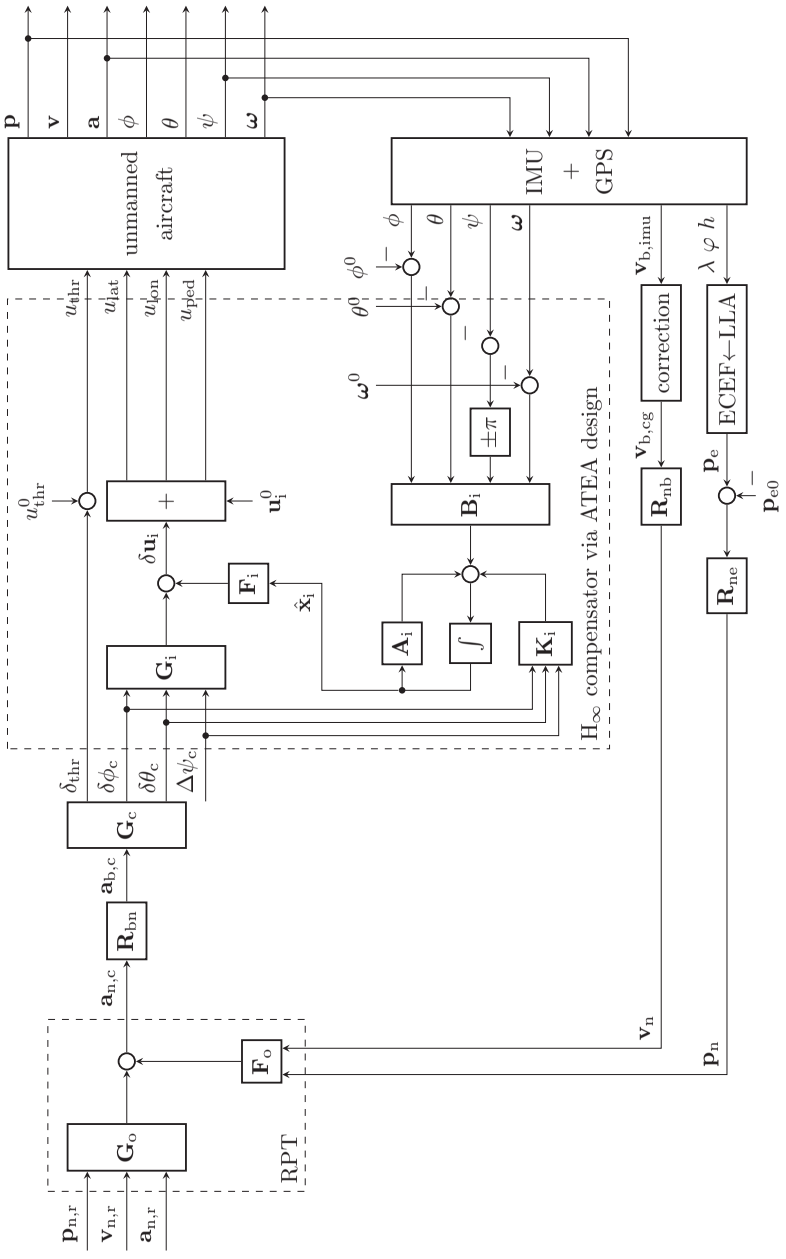


Figure 3.13: GremLion control structure with inner-loop and outer-loop

Based on the above control structure, the corresponding data flow diagram which fulfills the control requirements is shown in Fig. 3.14. In the *Task scheduling*, a control bubble *Mode selection* is deployed to realize the switch between semi-automatic control and fully automatic control. The switch is decided by the ground user command. By default, the GremLion is in semi-automatic mode when started. The semi-automatic mode can ease the manual operation of ground pilot and increase the stability of manual control. The GCS user can issue command "mode" to switch between two modes. In semi-automatic mode, the outer-loop reference is generated from the joystick of the RC controller. While in the fully automatic mode, the reference is generated from the path. The two modules are executed only one at a time dependent on the decision from the *Mode selection* and can be switched smoothly during the flight.

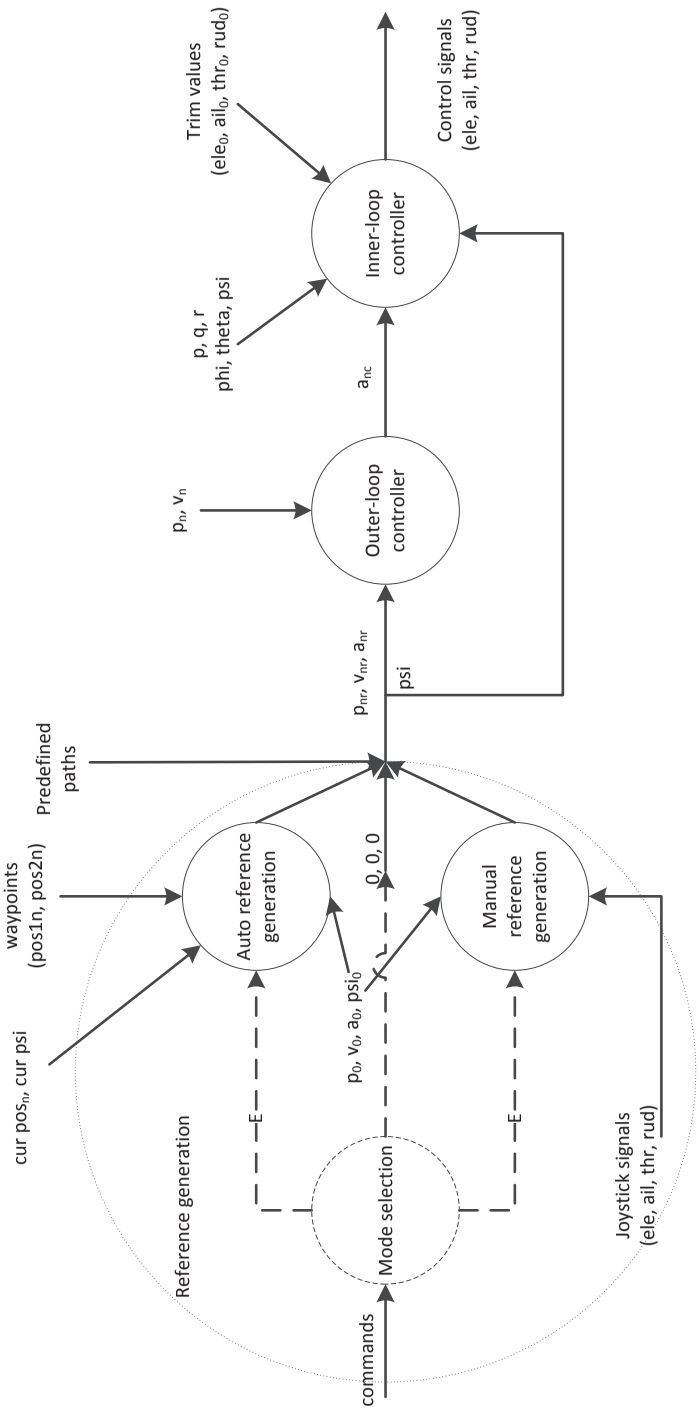


Figure 3.14: Data flow diagram of GremLion control structure

Table 3.4: Path structure

time	x	y	z	c
t_1	x_1	y_1	z_1	c_1
t_2	x_2	y_2	z_2	c_2
...	...	...	...	...
t_n	x_n	y_n	z_n	c_n

Example - HeLion, SheLion

The control structure for HeLion and SheLion also falls into the data flow presented in Fig. 3.14. For these twin UAV helicopters, no semi-auto mode is needed but still follows the same control structure in Fig. 3.10. As listed in Table 3.3, another set of outer-loop control law Bn() and inner-loop control law An() are activated, but with different control parameters. This modular design of control laws will reduce the coupling problems when deployed on multiple platforms simultaneously.

3.5.4 Path Generation

As shown in Fig. 3.10, the outer-loop references are needed to guide the UAV to perform automatic behaviors. In essence, all the automatic behaviors are conducted based on the outer-loop references. The outer-loop can also be regarded as the path generation phase which the desired position, velocity and acceleration can be specified at a time instant. The path is represented as a matrix with each column representing the time, positions, velocities and accelerations. The dimension of the matrix is determined by the given time points. Table 3.4 is an illustration of the path structure. The path can be represented in the matrix form with a dimension of n by m. The dimension n represents the time line for the path and m represents the number of reference variables in the control law.

The time line is displayed in the first column, where the time starts from 0. The other columns are the references of position, heading, velocity and acceleration. During the track-

Table 3.5: Autonomous path matrix construction - waypoint 0

time	x	y	z	c
0	x_0	y_0	z_0	c_0

ing process, the start time of start tracking is recorded and further reference is determined by the time elapsed from the start. A bisection search algorithm is deployed to realize the linear interpolation of the data from column 2 to column 5 given the current elapsed time duration.

3.5.5 Autonomous Reference Generation

In some critical applications, the reference paths for UAV must be generated online. This is commonly needed in dynamic environments such as in the cases of lost link, waypoints update from ground pilot. The data flow diagram of autonomous reference generation is shown Fig. 3.15. There are basically two parts, one is the path creation given a certain task, the other is the outer-loop reference generation from the generated path. In *PathCreation*, the task needs to provide the destination waypoint, given current position and heading, the new path can be automatically generated with a constant tracking velocity which can be specified. The destination waypoint can be uploaded from Google Maps with the GPS information. Alternatively, user can specify a destination point with relative distances based on the current starting position. Fig. 3.17 illustrates the path generation scenario. The coordinate system is based on the NED frame, when UAV starts onboard program, it records down the starting point O (lat_0, lon_0, alt_0). After some preliminary flight, the UAV reaches point S (x_0, y_0, z_0, c_0), where x_0, y_0, z_0 are calculated based on current GPS data and starting point GPS data. Then at this point, the user uploads the new waypoint A (lat_0, lon_0, alt_0). The new path is created in the following procedures:

1. Create the 1^{st} row of path matrix, the first track point should keep the current position and heading;

Table 3.6: Autonomous path matrix construction - waypoint 0, 1

time	x	y	z	c
0	x_0	y_0	z_0	c_0
t_1	x_0	y_0	z_0	c_1

Table 3.7: Autonomous path matrix construction - waypoint 0, 1, 2

time	x	y	z	c
0	x_0	y_0	z_0	c_0
t_1	x_0	y_0	z_0	c_1
t_2	dx	dy	z_0	c_1

2. Create the 2^{nd} row of path matrix. This track point is to change the heading angle to the desired value. The incremental change of the heading is derived via

$$\arctan\frac{dy}{dx} - c_0$$

where dx and dy are calculated from the current latitude and longitude.

Given the head turning rate 0.2 π rad/s, the time cost to finish the head turn operation is calculated via

$$t_1 = \frac{|\arctan\frac{dy}{dx} - c_0|}{0.2\pi}$$

Thus with the updated c_0 and t_1, the path matrix is updated as below:

3. Create the 3^{rd} row of path matrix. This track point is to perform the 2D path tracking given a constant velocity of 4.5 m/s. The relative distances x and y are as calculated above. The time duration for the 2D path tracking can be derived as:

$$t_2 = \sqrt{dx \times dx + dy \times dy}/4.5 + t_1$$

With updated t_2, dx and dy, the 3^{rd} point is listed in Table 3.7.

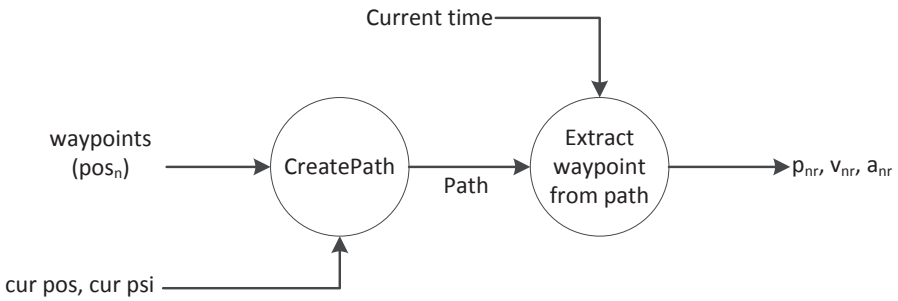


Figure 3.15: Data flow diagram - Autonomous reference generation

Finally, the autonomous one point path has been created and is fed into *ExtracWaypoint*, where the current reference of position, velocity and acceleration are derived. As the path tracking is designed based on time tracking, where the outer-loop reference is generated at an instant time based on the proportion of elapsed time with the whole path duration. For example, UAV starts to track path in Table 3.7 at time t_0. The first outer-loop reference will be

$$p_{nr} = [x_0, y_0, z_0, c_0], v_{nr} = [0, 0, 0], a_{nr} = [0, 0, 0]$$

The outer-loop reference derived at time t_n, $(t_1 < t_n <= t_2)$, will be

$$p_{nr}[0] = \frac{(t_2 - \Delta t)}{t_2 - t_1} \times x_0 + \frac{(\Delta t) - t_1}{t_2 - t_1} \times dx$$

where

$$\Delta t = t_n - t_0$$

, which represents the elapsed tracking time from starting time t_0. This linear interpolation method also applies to $p_{nr}[1]$ and $p_{nr}[2]$, while the heading angle reference $p_{nr}[3] = c_1$. On the other hand, the velocity reference p_{vr} can be calculated via numerical differentiation.

However, the derived position and velocity have a step change at the beginning of the path and end of the path as well. This will challenge the dynamics of the UAV regarding

the control stability. A better way is to refine the path points such that there is a transient change in position and velocity during the whole path tracking period. For example, the cruise flight speed is set as 2 m/s in the 2D path tracking. Instead of commanding the UAV to increase speed directly from 0 m/s to 2 m/s, the position reference can be appended with extra path points each with an incremental distance of 0.5 m per second. Below is the code snippet which realizes the path matrix points appending at the beginning phase of the path:

```
// append 1st point
t += 1;
m_path[n][0] = t;
m_path[n][1] = m_path[n-1][1] + 0.5*cos(c1);
m_path[n][2] = m_path[n-1][2] + 0.5*sin(c1);
m_path[n][3] = m_path[n-1][3];
m_path[n][4] = c1;
n++;

// append 2nd point
t += 1;
m_path[n][0] = t;
m_path[n][1] = m_path[n-1][1] + 1*cos(c1);
m_path[n][2] = m_path[n-1][2] + 1*sin(c1);
m_path[n][3] = m_path[n-1][3];
m_path[n][4] = c1;
n++;

// append 3rd point
t += 1;
m_path[n][0] = t;
m_path[n][1] = m_path[n-1][1] + 1.5*cos(c1);
m_path[n][2] = m_path[n-1][2] + 1.5*sin(c1);
m_path[n][3] = m_path[n-1][3];
m_path[n][4] = c1;
n++;
```

Similarly, the same approach can be applied at the end of the path. The final generated

path with refinements is shown in Fig. 3.16. In this task, the UAV is commanded to fly forward 20 m with a cruise speed of 2 m/s. ug and vg represents the practically forward velocity and lateral velocity in NED frame, respectively, while $ug - ref$ and $vg - ref$ represents the velocity reference. It is clear that both $ug - ref$ and $vg - ref$ are smoothed with extra six points.

As the onboard main loop execution is running at 50 Hz, the outer-loop reference also needs to be updated every 0.02 ms. Therefore, the path tracking elapsed time is accumulated with an incremental time of 0.02 ms.

3.5.6 Emergency Precaution

This additional safety measure is a result that we learned from a couple of crash accidents. There are many sources that would cause failures in the UAV control system, which include drastic changes in environment, hardware failure, GPS disorder and problems in control system design and software implementation. A mechanism for handling emergency situations is built into the onboard software system. At every cycle, before applying control action, the control task thread checks all data received from the IMU and other sensors. Once any abnormality is observed, the emergency control function is activated immediately to sequentially 1) send an alert signal to the ground control station to inform the pilot to take over the control authority, 2) drive and maintain all the input channels to their trimmed values in the hovering condition to stabilize UAV attitude.

3.5.7 Onboard Configuration File

To make the overall software system more flexible, an onboard configuration method is designed to fulfill the different requirements of each UAV platform. Each UAV platform has different hardware configurations with various peripheral sensors. Also, each UAV has its own specific control parameters even though they share the same control structures. As such, all these configuration related parameters are stored in a local file. When the onboard

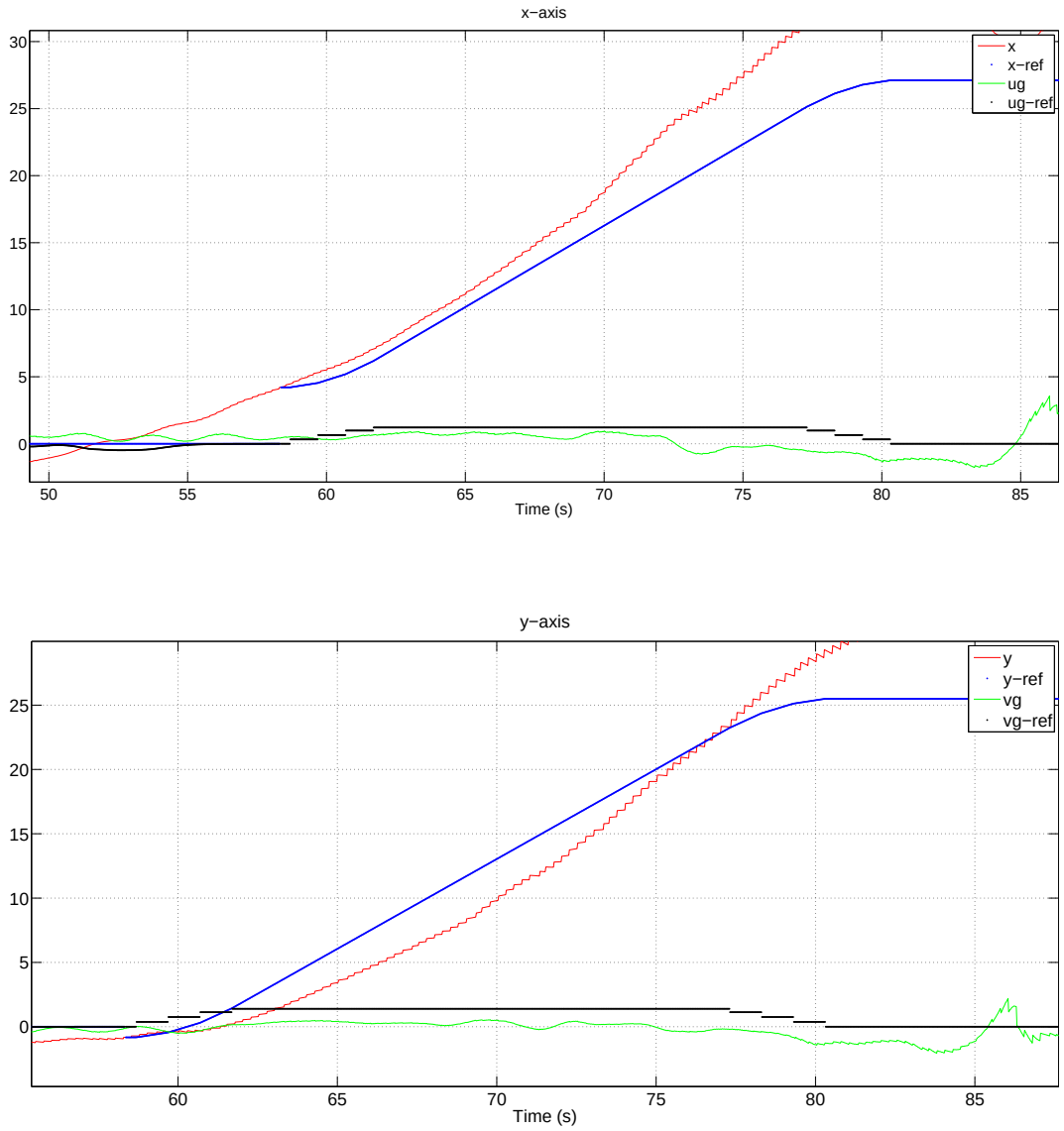


Figure 3.16: Refined outer-loop reference

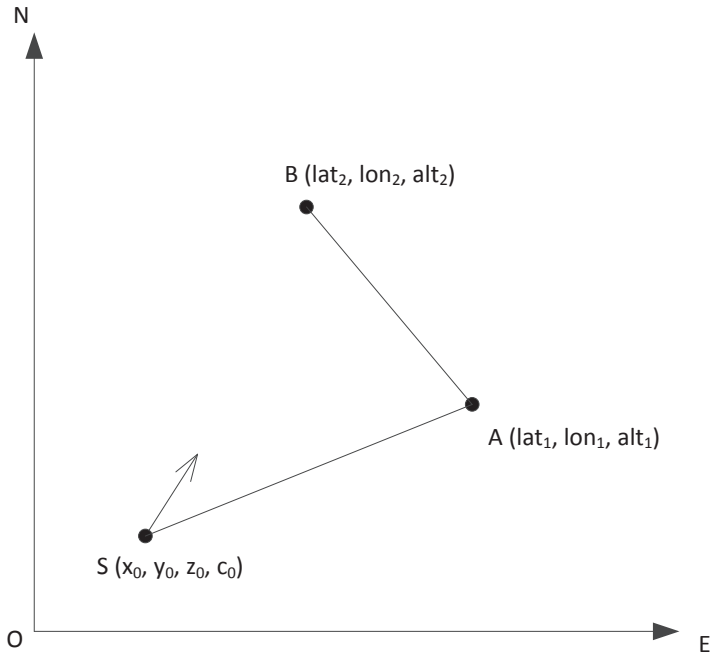


Figure 3.17: Autonomous path generation scenario

program starts to execute, it first loads all the parameters given a specific platform. During the onboard program execution, the parameters are called accordingly.

Below is an illustration of onboard configuration file for GremLion. The *_HELICOPTER* 11 denotes the GremLion ID of the current UAV platform. Other global parameters such as gravity of the earth given different locations is stored in *_gravity*. All these parameters are parsed initially and loaded as necessary parameters such as control matrices **F** and **G** in the outer-loop and inner-loop. For example, the linear model for GremLion with matrices **A** of order 11 by 11 and **B** of order 11 by 4 is also provided in the configuration file.

```
_HELICOPTER = 11;
_gravity = 9.851;
_m_A_GREMLION = [
   -0.0508    0      0      0    0   -9.8100   -9.8100    0    0    0    0;
         0   -0.0500 0      0    9.8100    0    0    9.8100    0    0    0;
         0   0       0      0    0    0   60.4987  203.0711     0    0     0;
         0   0       0      0    0    0  134.0256  -89.4702     0    0    0;
```

```
        0   0       1.0000 0    0     0    0   0    0     0       0;
        0   0       0      1.0000    0     0    0   0    0     0       0;
        0   0       0      -0.4281    0     0  -10.3461   -1.6466    0    0    0;
        0   0      -0.4281   0    0    0    4.8324  -10.3461    0    0     0;
        0   0       0      0   0    0    0    0   -0.3230    0.1161    0;
        0   0       0      0   0    0    0    0    0  -16.4702        0;
        0   0       0      0   0    0    0    0    0    1.0000         0
];
_m_B_GREMLION = [
        0         0         0         0;
        0         0         0         0;
        0         0         0         0;
        0         0         0         0;
        0         0         0         0;
        0         0         0         0;
    0.4499    1.4488         0         0;
   -1.1825    0.1135         0         0;
        0         0  -11.9686   -8.0761;
        0         0         0 -281.4877;
        0         0         0         0
];


Outerloop_F = [
     -0.2961   0        0        -1.2566  0         0;
     0         -0.2961  0        0        -1.2566   0;
     0         0        -0.7402  0         0      -2.0420
];
Outerloop_G = [
    0.2961         0         0;
         0    0.2961         0;
         0         0    0.7402
];
Innerloop_F = [
   -0.0033 -0.0097 0.4087 -0.2385 1.9084 -0.6430 1.3846  8.7457   0   0   0;
   -0.0048 0.0067 -0.4368 -0.1143 1.3158 -0.9326 -5.0959 -4.4746  0   0   0;
    0      0     0      0     0     0     0     0     0     0    0;
    0      0     0      0     0     0     0     0     0     0    0.0014 0.0224
];
Innerloop_G = [
```

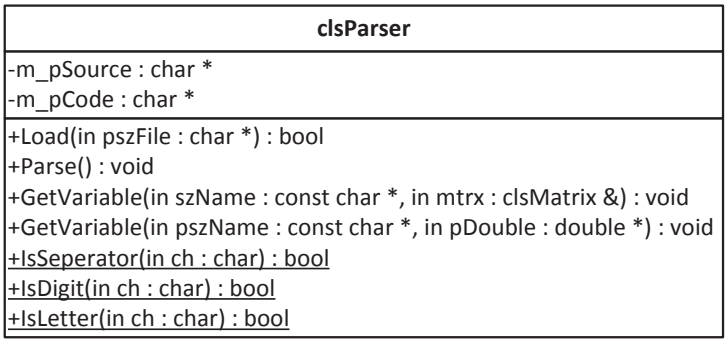


Figure 3.18: Class diagram - clsParser

```
   -0.0655   -0.1945 0     0;
   -0.0951    0.1341    0     0;
   0           0 0     0;
   0       0   0 -0.0224
];
```

In order to parse the variables and corresponding matrix values in the file, an assistant class *clsParser* is developed to extract the parameter and values. Fig. 3.18 lists the variables and functions implemented in the *clsParser*.

3.6 Servo Driving

3.6.1 Logical Representation

The servo driving block is to realize three functions: 1) Send request to servo board for manual data; 2) Receive and record the manual control signals from ground pilot; 3) Output automatic control signals calculated from the *Flight control system* to the actuators. The servo driving board adopted in the onboard control system is UAV100 from PONTECH company [59]. The manual data refers to the radio control signal from the ground transmitter operated by a professional pilot. The manual data from ground pilot is used for UAV modeling purpose. It is necessary to record down for post-flight manual flight analy-

sis. Based on the data sheet, the hardware communication protocol requires the onboard software system to send a request to the servo before it can receive the manual data. The ground pilot has the final authority to determine whether manual signal or automatic signal is passed to the actuators.

3.6.2 Software Modeling

Based on the requirements above, the data flow diagram of *Servo driving* is shown in Fig. 3.19. There are two incoming data as explained above. One is the manual control signal from the ground pilot, the other one is the automatic control signals. Based on the switch signal, the data flow destination of the two incoming data is decided based on the control transformation *Decide output channel*. If the manual control is enabled, then the manual input signal will be applied to the actual control of UAV. Otherwise, the automatic signal will be sent to realize automatic flight control. Fig. 3.20 is the class diagram of the servo driving task, which lists all the servo functions with variables and methods to fulfill the servo driving specifications. The numbers within the data flow diagram indicate the bubble execution sequence to resolve the scheduling situation where there are three parallel data flows in the *Servo driving*.

3.7 Data Logging

3.7.1 Logical Representation

Another important block for onboard system is the data logging, though not displayed in Fig. 2.2. The data logging block is to record all the data existed in the whole system, such as the UAV status data from sensing, manual and automatic servo signals in the servo driving block, and user commands from GCS. As discussed in the above sections, each block will generate internal data flows which will reflect the working status and property of the whole system. Considering the CPU load increases when writing data to the onboard CF card,

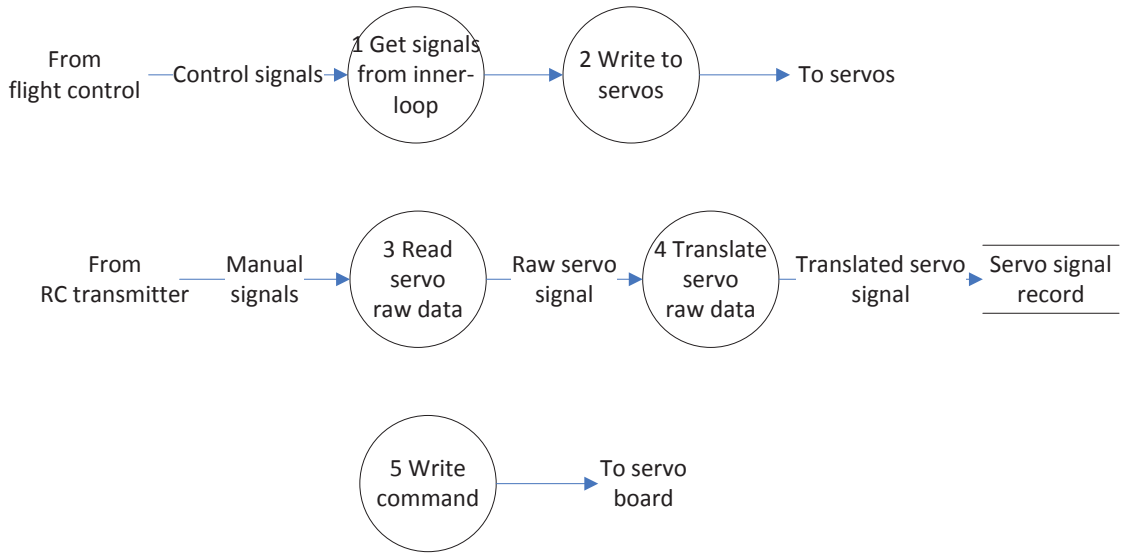


Figure 3.19: Data flow diagram - Servo driving

clsSVO
-m_nsSVO : int -m_tSVO : double -m_sig : UAVCONTROLSIG -m_svo : SVODATA -m_svoRaw : SVORAWDATA -m_tSVO[MAX_SVO] : double -m_svoRaw[MAX_SVO] : SVORAWDATA -m_svo[MAX_SVO] : SVODATA
+GetSig() : void +WriteCommand() : void +GetData(in psvoRaw : SVORAWDATA *) : SVORAWDATA +Translate(in psvoRaw : SVORAWDATA *, in psvo : SVODATA *) : void +SetRudder(in psig : UAVCONTROLSIG *) : void +InitThread() : void +EveryRun() : int +ExitThread() : void +GetSVOData() : SVODATA &

Figure 3.20: Class diagram - clsSVO

the data logging thread is executed every 50 cycle, *i.e.*, once per second.

It is also very important to keep logging data as much as possible on emergency situations, which can be used to identify problems causing the incidents. However, in an exceptional occasion when a crash occurs, all onboard tasks executing including data logging are not terminated normally. In a normal operation, the data logging file is opened at the beginning of a flight test, and is appended with in-flight data during the test, and finally closed and saved at the end of the test. In an abnormal situation, the data logging file is likely to be corrupted, resulted in losing all record data. If the process is interrupted, the whole file will be damaged. A mechanism similar to the black box in commercial aircraft is implemented to keep saving logged data even in a crash. Such a feature is illustrated in the program below:

initialization stage

do nothing

log stage

1st time (the first 50 cycles)

open log file in writing and appending mode

write 1st pack of data to log file

close log file

2nd time (the next 50 cycles)

open log file in writing and appending mode

write 2nd pack of data to log file

close log file

...

n-th time

open log file in writing and appending mode

write n-th pack of data to log file

close log file

...

final stage

do nothing

As illustrated in the program above, the new feature enables the data logging process to complete data writing and saving in every 50 cycles. If the onboard system halts due to crash or out of power, the flight status data can be traced back to the last second which still can provide enough data for analyzing the accident.

3.7.2 Software Modeling

The modeling of the above pseudo code is shown in Fig. 3.21. As shown in the figure, all the record data previously stored in each task thread, such as IMU, DAQ, CTL, SVO are dumped into the CF card via the DLG records by shared memory copy operation. Based on the requirements described in the data flow diagram, the corresponding class diagram is shown in Fig. 3.22.

3.8 Task Identification

Given the detailed level 2 DFD, the task grouping and identification can be realized as shown in Fig. 3.23, in which all major functionalities of the onboard system are provided. The task grouping criteria here is based on the hardware dependency principle. For example, the sensing hardware IMU with GPS is grouped into one single task IMU. The servo driving functions are grouped as another task SVO. The data acquisition board for retrieving ultrasonic sensor is identified as DAQ. The working thread NET is designated for sending UAV to UAV data while CMM is designed for air to ground data transmission. The control

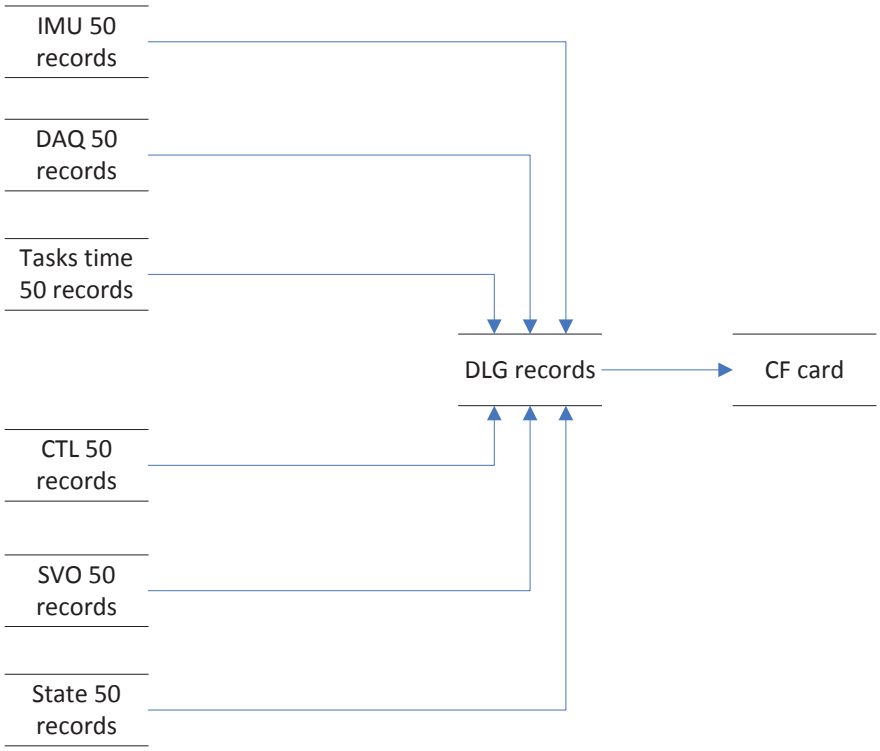


Figure 3.21: Data flow diagram - Data logging

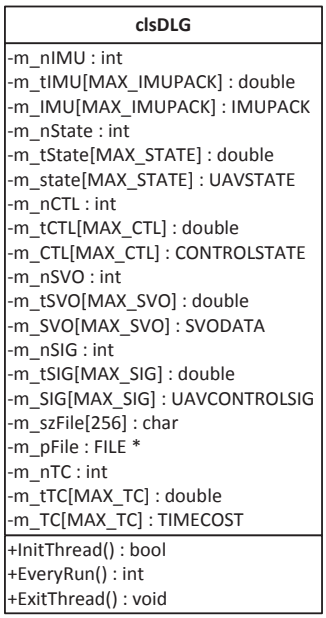


Figure 3.22: Class diagram - clsDLG

algorithms related functions are grouped as CTL thread. The last working thread DLG is for data logging related work.

3.9 Task Management

Given the identified working threads in Fig. 3.23, the software architecture should be carefully designed to fulfill the flight control critical task. A multiple-threads architecture is deployed in QNX Neutrino. The QNX provides the kernel level mechanism to support the message passing and synchronization of multi-thread software architecture [30, 37]. The library functions used for task messaging and synchronization are listed in the Table 3.8 for reference.

Each identified task is implemented in a separate thread given the assigned CPU execution time. For onboard flight control processor, the tasks should be executed in the designed order: retrieve sensor data (navigation data from IMU (*clsIMU*), ultrasonic sensor

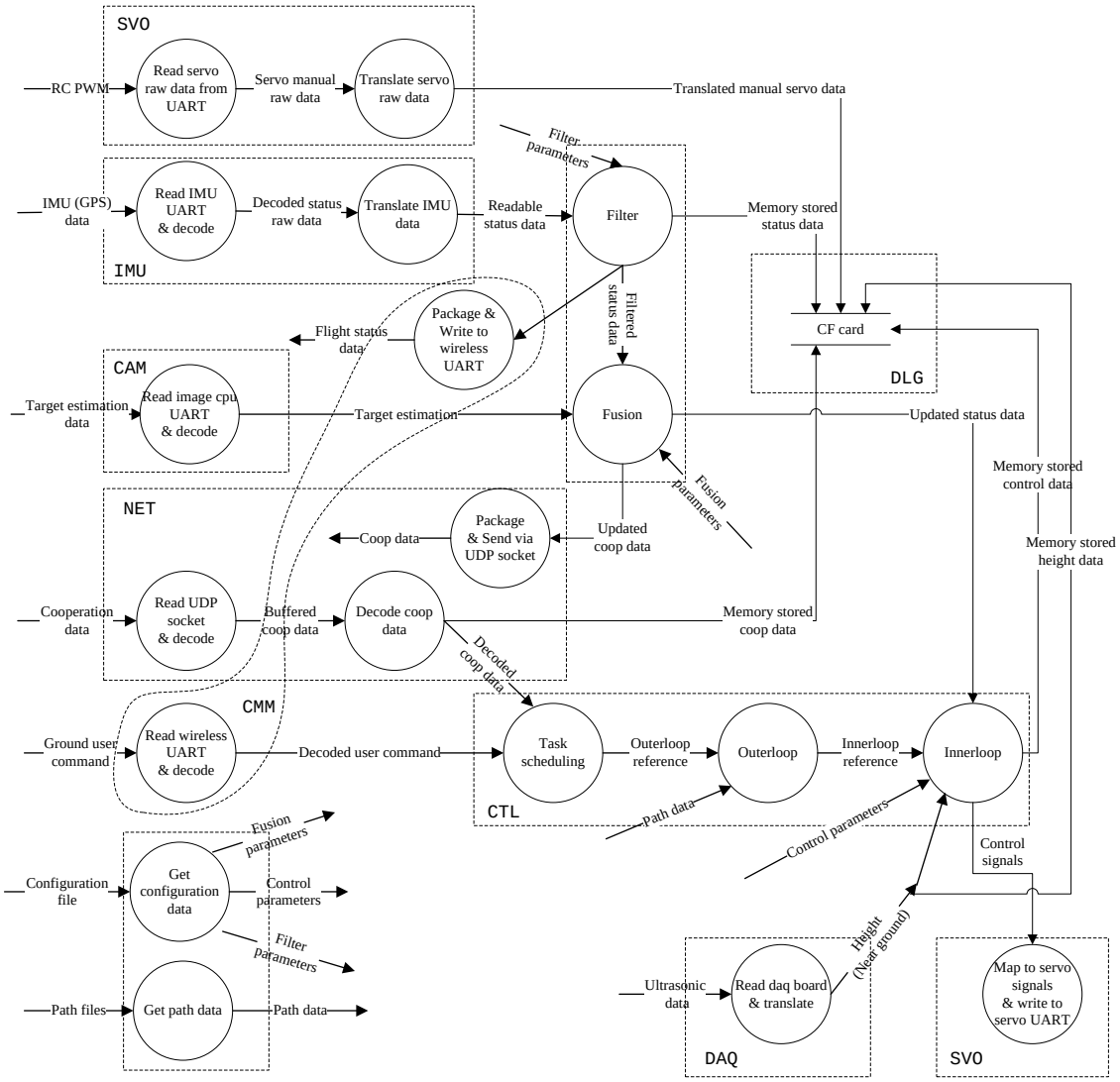


Figure 3.23: Data flow diagram with task identification

Table 3.8: QNX run-time functions

Function	Description
pthread_create	Create a new thread
pthread_mutex_lock	Lock a mutex object
pthread_mutex_unlock	Unlock a mutex object
pthread_cond_signal	Signal a condition variable and unblock threads waiting for it
pthread_cond_wait	Wait for a condition variable and block the calling thread on it
timer_create	Create a timer
timer_settime	Set property of a timer (starting time, repeating interval)
MsgSendPulse	Send a pulse to a channel
MsgReceivePulse	Receive a pulse from a channel
TimerTimeout	Set expiration time for waiting function

(*clsDAQ*)) and get vision information from camera subsystem (*clsCAM*), execute control law (*clsCTL*), drive sensors (*clsSVO*), send communications to GCS (*clsCMM*) and record log data to onboard storage card (*clsDLG*). The onboard program is designed to execute every $20ms$ to fulfill the required control frequency. Each loop will run the task threads *clsIMU*, *clsDAQ*, *clsCTL* and *clsSVO*. While the tasks threads *clsCMM* and *clsDLG* are scheduled to run every 50 loops, i.e. 1 Hz to improve CPU efficiency. Fig. 3.24 illustrates the task scheduling approach. A special task *main* is designed as the task scheduler to manage all the task threads. As shown in the figure, the continuous main execution loop is represented with one horizontal CPU execution time period. The timer is set to send an impulse to the scheduler every 20 ms, once the scheduler program receives the impulse, it starts scheduling the execution of each thread. The task synchronization mechanism is realized by the activation/notification method provided by the QNX kernel function calls.

The grey area represents the actual execution for each task, while the blank area denotes CPU idle time. Fig. 3.25 shows the message interaction when performing the task synchronization. First, the scheduler and *clsIMU* are both in idle state. Once scheduler transfers to active state when the timer signal arrives, it sends *MsgSendPulse* message to task *clsIMU*,

then the task *clsIMU* is activated and starts executing with an assigned time allocation of T_{IMU}. Meanwhile, the scheduler transfers to the *pthread_cond_wait* state. It keeps waiting until the task clsIMU sends the *pthread_cond_signal* message to it. Next, the scheduler will perform the same task synchronization for the following tasks *clsDAQ*, *clsCAM*, *clsCTL*, *clsSVO*, *clsCMM(TX)*, *clsDLG* until all tasks are finished. Please note there is also another task thread *clsCMM(RX)* which denotes a background task to receive user commands from GCS. It is scheduled to be activated whenever CPU is idle by the QNX scheduler. It is because the receiving task is needed to be executed only when the data arrives instead of keeping activated periodically. For the task *clsCMM*, the maximum bandwidth of the wireless modem is 115200 bit per second, i.e., it takes about one second to transfer 14 KB (kilo bytes) of data. Considering the sending device is relatively a slow processing unit that cannot afford massive data transferring, the task thread *clsCMM(TX)* is scheduled to execute once every 50 loops. Given task scheduling requirements, the round-robin scheduling policy in QNX is selected for flight control. With round-robin, a working thread continues executing until 1) voluntarily relinquishes control, 2) is preempted by a higher-priority thread, 3) consumes its assigned timeslice.

The pseudo code of the main program is described as below:

main program

initialize task threads

initialize timer

loop {

wait for timer signal

read user command

if command is for exit, exit loop

send an activating pulse to task thread 1

wait some time for task 1 accomplishment

send an activating pulse to task thread 2

wait some time for task 2 accomplishment

...

send an activating pulse to task thread n,

wait some time for task n accomplishment

}

send an exit pulse to task thread 1,

send an exit pulse to task thread 2,

...

send an exit pulse to task thread n.

exit main program

task thread

loop {

wait for a pulse

if pulse is for exit, exit loop

do work

...

set notification of accomplishment

}

exit task thread

Assessment of time consumption for software and hardware is performed by testing functions in the program. For a specific task thread, the beginning time and end time are recorded using timer function. Statistical data are logged for all tasks. From the recorded time data, the time consumption allocation for each can be assigned. Various IMU, servos and data acquisition board takes different time to send back sensor data. As such, the time

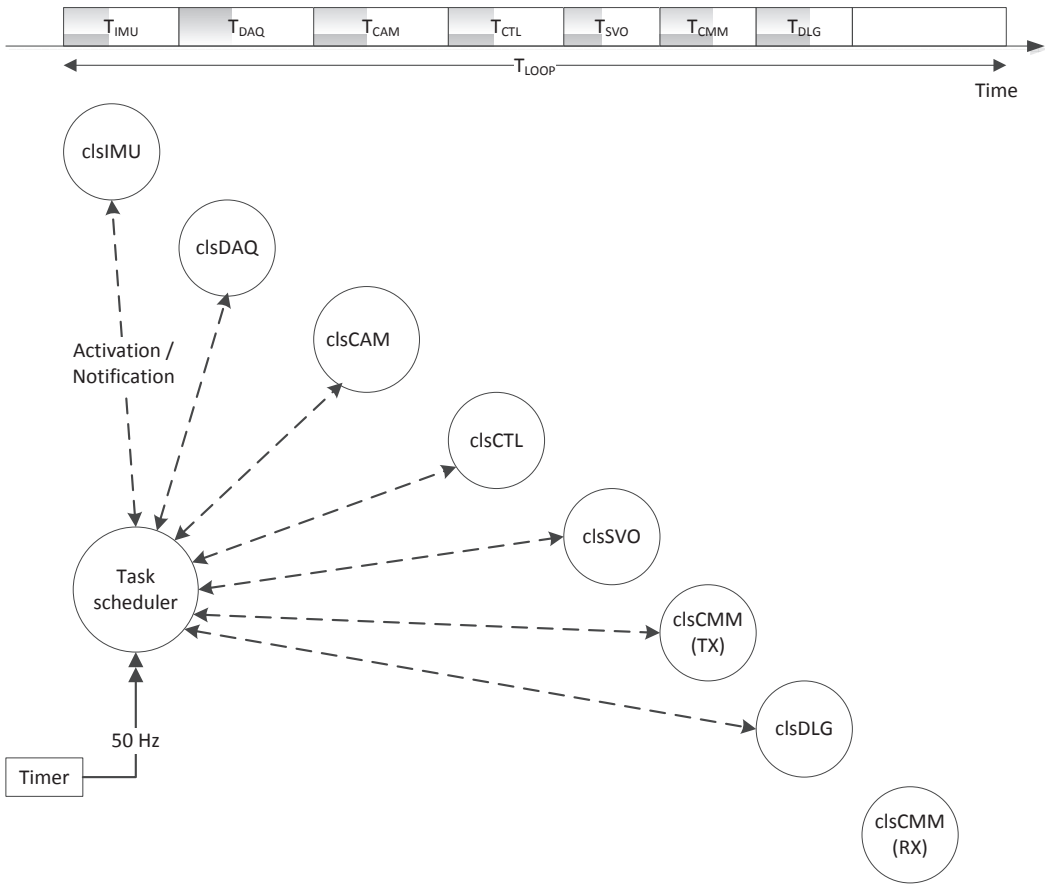
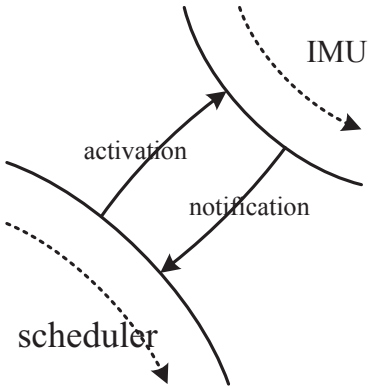


Figure 3.24: Onboard task management

allocation for each task is more than the actual time needed to ensure the hardware data is ready to be received in each main loop. On the other hand, the sum of time allocation for each task should not exceed the main loop execution time 20 ms to maintain the 50 Hz control update rate. For example, the time allocation for each task on control processor of GremLion is listed in Table 3.9.

After identifying task threads and management requirements, we need to model the task thread behaviors. It is clear that the task synchronization approach is the same for each working thread. A class *clsThread* is designed as a parent class for all the task threads. The *clsThread* can be inherited to integrate more working threads which facilitates the



MsgSendPulse
MsgReceivePulse
pthread_cond_signal
pthread_cond_wait

Figure 3.25: QNX task synchronization approach

Table 3.9: Time allocation for each task thread on the control processor

IMU	DAQ	CTL	SVO	CMM	CAM	DLG
2 ms	1 ms	1 ms	2 ms	1 ms	1 ms	5 ms

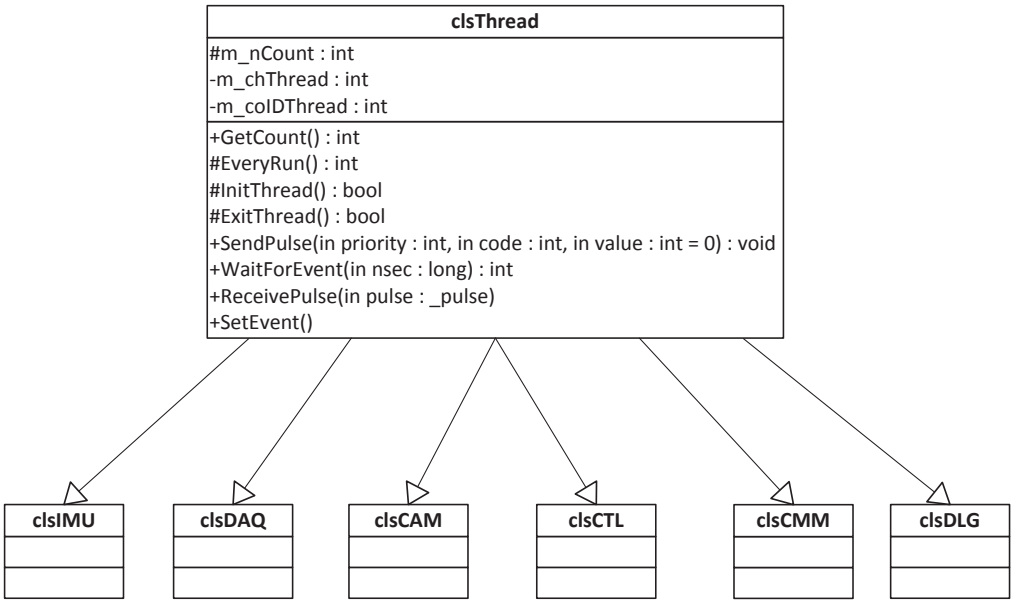


Figure 3.26: Class diagram of onboard task threads

implementation of new hardware modules.

3.10 System Behavior Modeling

After the software modeling of system tasks, like the properties and functions in each class, the dynamic behaviors of the system are ready to be described via the sequence diagram. The sequence diagram is to describe the sequential arrangements of the data flows among the collaborating class objects. An automatic task completion is via the individual object functions and the inter-objects collaboration as well. Therefore, with a clear sequence diagram description, the system level behaviors of the onboard system will be presented in a time-line approach. Fig 3.27 is the activity diagram when the hardware-in-the-loop simulation task is executed. As shown in the figure, the class objects *clsMain::_main*, *clsCMM::_cmm* are involved in the task. The dotted vertical line below each object represents the object life-time. Only one instance for each class is deployed in the onboard system, thus the functions

of each object will keep active until the onboard program exits. The solid arrow represents the function calls from one object to another object.

The onboard task scheduler thread will start execution first. The object *_main* will call the function of *_cmm* to retrieve the commands received from GCS. If any user command is received, *_main* will call the function *PutCommand* of the object *_ctl*. In each loop execution of the control task, *_ctl* will call the function of *GetSimulationType* of the object *_state*. Given the simulation type, for example, in this case, model-based simulation task is selected. Thus, the corresponding model of this platform is called in the function *Simulate*. Once the model finishes the simulation iteration, the control task thread will examine the received user command via its own functions *GetCommand*, extract out the behaviors and parameter via the function *ProcessCommand*, then set behavior to the UAV, e.g. "hold", "path tracking". Then the *ReferenceGeneration* will be called to derive the outer-loop references for the outer-loop controller *Bi*, whose output is fed into inner-loop controller *Ai*. Both *Bi* and *Ai* will call the function *GetState* of *_state*. Once both control laws are executed, *_ctl* will call *UpdateSIG* from *_state*. Finally, the servo task thread object *_svo* will set the control signals to the servo boards.

3.11 Onboard Vision Subsystem

3.11.1 Introduction

With the autonomous control capability, advanced intelligence can be achieved with the information-rich vision system. Therefore a vision subsystem is exclusively developed to explore the vision advantages. The onboard vision system can take live pictures via onboard digital camera, followed by vision related algorithms which can assist the onboard high level intelligent control usage to achieve advanced control behaviors such as obstacle avoidance, ground target detection and tracking. Given the task requirements of vision system, the onboard vision data flow diagrams of different levels are shown from Fig. 3.28 to Fig. 3.29.

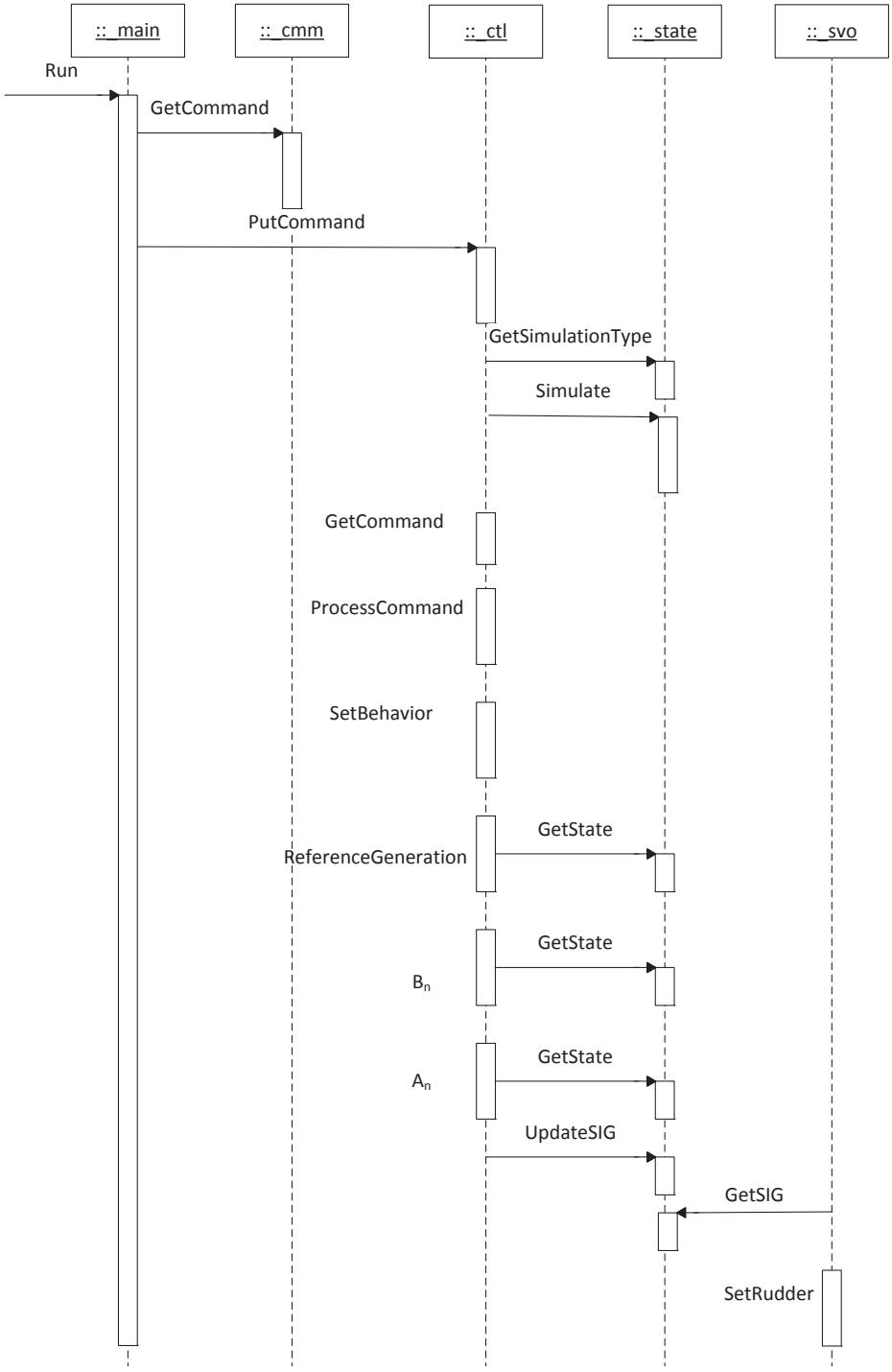


Figure 3.27: Sequence diagram - hardware-in-the-loop simulation

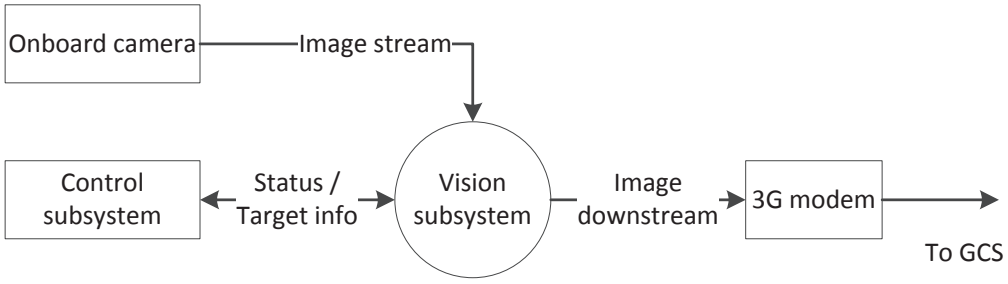


Figure 3.28: Data flow diagram of onboard vision software system - Level 1

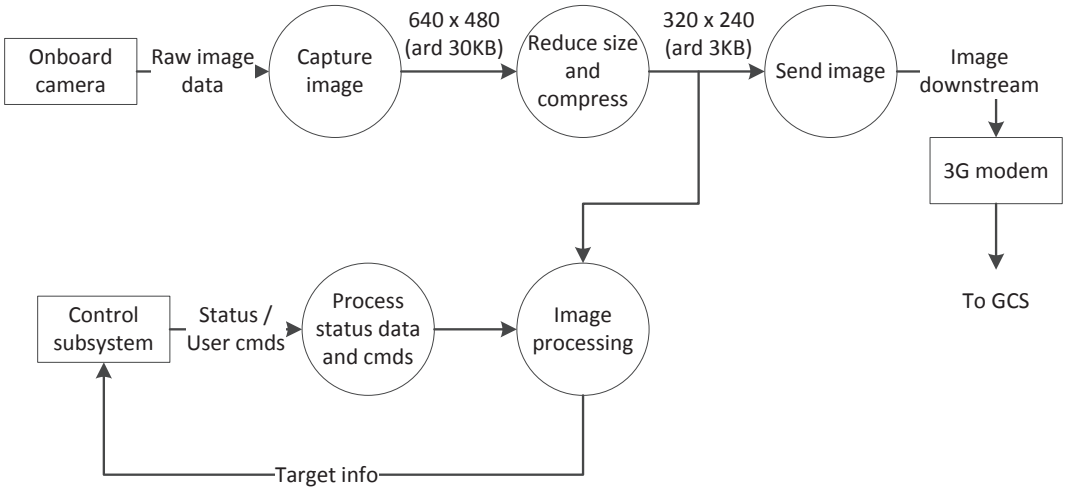


Figure 3.29: Data flow diagram of onboard vision software system - Level 2

3.11.2 Task Management of Vision Subsystem

Though the operating system on vision subsystem is different from the one on the control subsystem, the task management of working threads is quite similar in the sense that all working threads are coordinated by a task scheduler to coordinate the execution sequence of each onboard tasks. The working principle is as follows, capture raw images from digital camera, receive the UAV status data from onboard control system, perform image processing including preprocessing and algorithms calculation, send compressed image over 3G modem. However, because both image capturing and image processing are computation intensive task threads, the main loop of vision subsystem is set as 10 Hz. As the vision algorithms is for high level guidance and control for the control subsystem, the 10 Hz update rate is acceptable in most applications.

3.12 Software Integration

The idea of software integration is to realize online configuration and module loadings given different UAV and even UGV (Unmanned Ground Vehicles) platforms. Based on the software framework designed in Fig. 2.2 which is universal to all automatic onboard systems, the software module developed in the previous sections can be reused by minor modifications to accommodate different UAV platforms developed in our group. Different onboard systems have various hardware components and different control laws, thus a unified software approach is necessary under the proposed framework. The design approach is like a plug-and-play, which the desired modules are loaded given different configurations. For example, SheLion UAV has an onboard camera, thus the camera thread is activated during the start of the application. Also, as the IMU sensors are different on HeLion and SheLion, specific sensor reading modules are loaded correspondingly. The approach can be described in the following pseudo code:

main initialization

read onboard configuration file

parse UAV ID numbers

case UAV_1*:*

read $sensor_1$*;*

execute $control_1$*;*

write to $servo_1$*;*

read / write communications $module_1$*;*

...

case UAV_n*:*

read $sensor_n$*;*

execute $control_n$*;*

write to $servo_n$*;*

read / write communications $module_n$*;*

...

All the module developed onboard are controlled by a flag. If the flag is on, then this module will be activated and executed during the program. Once the UAV ID is determined, the flags for the task thread are also determined by the predefined UAV configuration file.

Fig. 3.30 demonstrates the onboard software modules. Given a specific UAV ID, the modules for this UAV are determined and thus deployed to execute during the whole onboard application.

3.13 Performance Evaluation

3.13.1 Task Timing

The time deadline is the most important property in real-time systems. As the control loop period of the control subsystem is set to 20ms, the timing intervals between each running

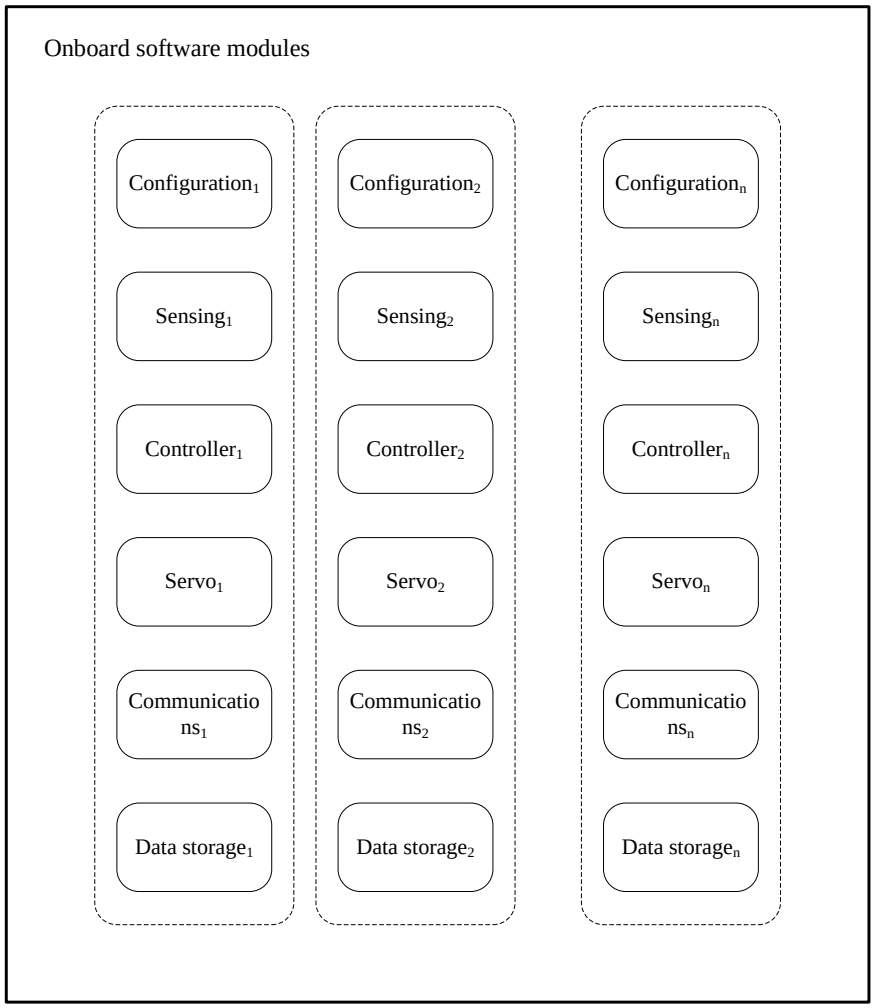


Figure 3.30: Onboard software modules

loop should be examined to test the robustness and efficiency of the complete real-time sofware system. Below is the summary of the task timings recorded in QNX. The time consumptions for each task threads are summarized in Fig. 3.31.

On the other hand, the total processing time for each loop is summarized in Fig. 3.32. It is obvious that the timing interval between each loop is separated by strictly following the 20ms configuration which provides the fundamental support for correct and stable control law implementation.

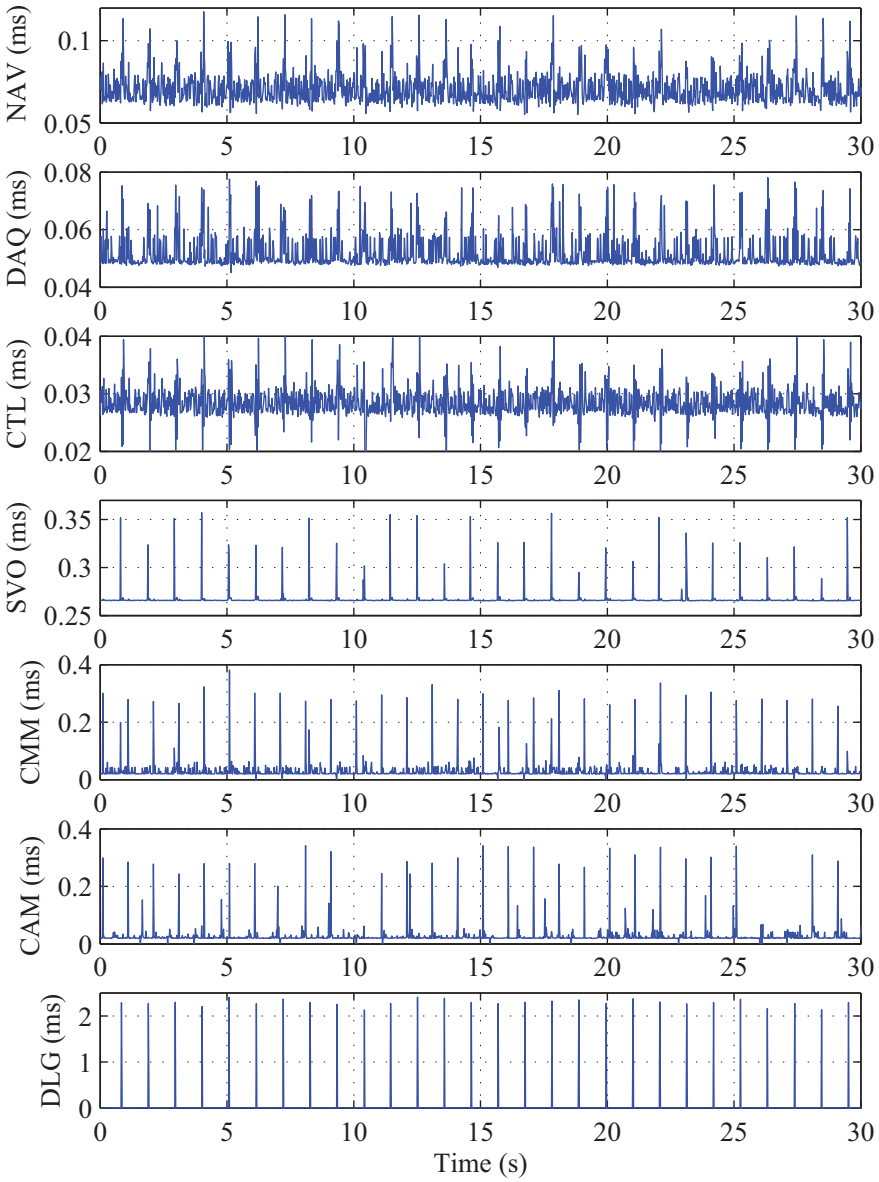


Figure 3.31: Time consumption statistics for each task thread

The CPU usage rate of the flight control computer can be calculated as the ratio of the total time consumption and the length of period, which is 20 ms used in the actual test. From the result shown in Fig. 3.33, we can determine that the lowest usage rate of the flight

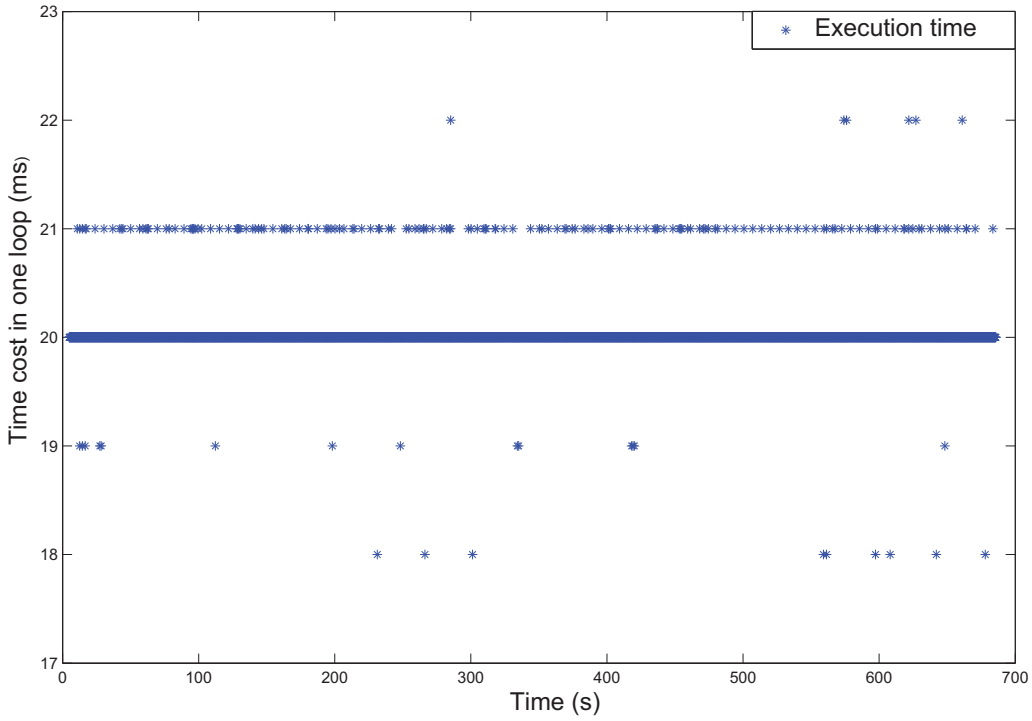


Figure 3.32: Time intervals between each loop

control computer is

$$\text{usage}_{\text{min}} = \tau_{\text{min}}/T_{\text{cyc}} = 0.45/20 = 2.25\%$$

and the highest peak is

$$\text{usage}_{\text{max}} = \tau_{\text{max}}/T_{\text{cyc}} = 2.93/20 = 14.65\%$$

where τ_{min} and τ_{max} stand respectively for the minimum and maximum total time consumptions by all threads in one cycle, and T_{cyc} is the 20 ms period of the cycle. The peak of the CPU usage rate comes with the large working load of data logging and communications, which occurs once every 50 cycles, *i.e.*, 1 second. The average time consumption in each

cycle is about 0.49 ms. Thus, the average CPU usage rate is

$$\text{usage}_{\text{ave}} = \tau_{\text{ave}}/T = 0.49/20 = 2.45\%$$

The low usage rate of the flight control computer is ideal for guaranteeing the reliability of the overall software system. To balance the working load, the communication and data logging threads have to be scheduled in a more distributive and efficient way.

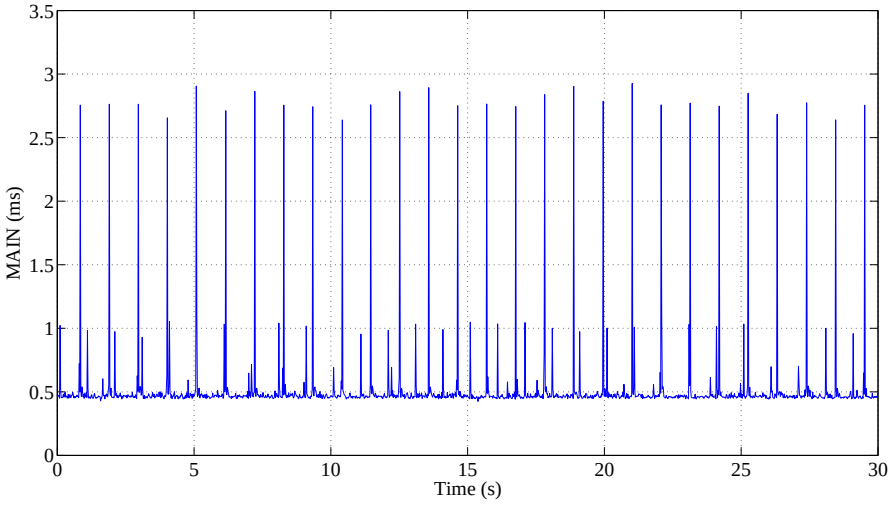


Figure 3.33: Time consumption statistics for main loop.

3.13.2 Multi-thread Reliability Test

The reliability test refers to the situation when one task thread blocks due to hardware malfunction, the other task threads should be able to continue execution without degrading the overall performance too much. For example, if the IMU task thread blocks during the reading operation, then the CTL task can still be activated when the timer assigned to IMU expires and the subsequent SVO task also executes to stabilize the UAV servos. On the contrary, if one task blocks lead to the whole onboard program halts, then the UAV will

render undetermined behaviors even an unacceptable crash.

To test the reliability of multi-thread architecture, the connection of IMU with processor is intentionally disconnected during the ground test. As we can see from the recorded data, the IMU data blocks record is less than other task data records. From time 220.1 second, the IMU hardware got blocked and no more data received in the program, the onboard main program can still successfully schedule the execution of other task threads. It is clear that the automatic control signals is still outputting to the servo board as shown from time 220.1 second beyond.

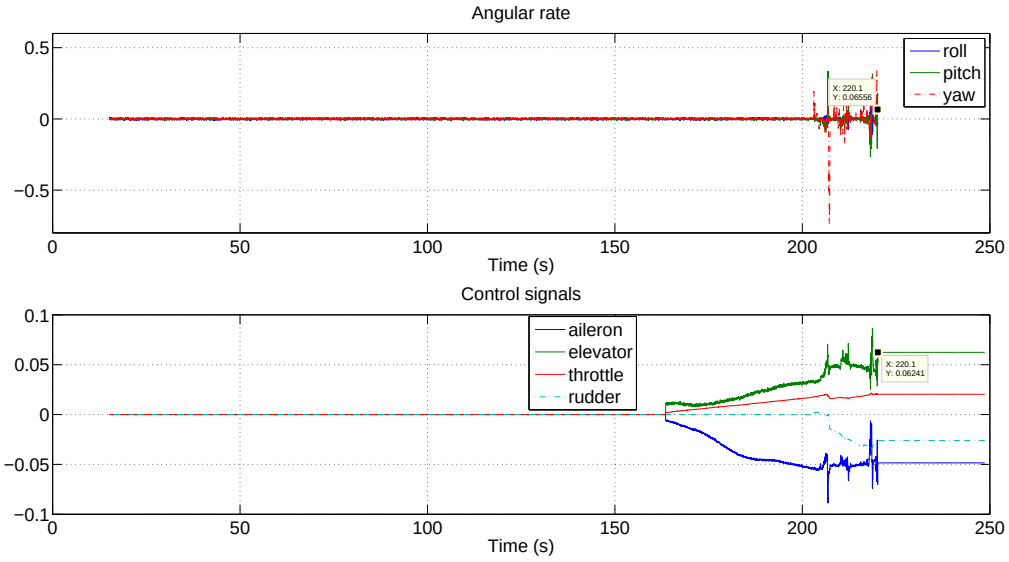


Figure 3.34: Multi-thread reliabilty test

3.13.3 Emergency Test

Due to the safety considerations, limitations for the flight status such as position, velocity, angular rate are defined in Table 3.10. The positions x, y and z are largely dependent on applications. In routine flight tests, the maximum flight distance will be controlled within line-of-sight range. While in applications such as competitions, the flight distance

Table 3.10: Flight safety limitations

x	[-1000m, 1000m]
y	[-1000m, 1000m]
z	[-100, 100m]
u	[-5m/s, 15m/s]
v	[-8m/s, 8m/s]
w	[-5m/s, 5m/s]
p	[-2rad/s, 2rad/s]
q	[-2rad/s, 2rad/s]
r	[-2rad/s, 2rad/s]
a	[$-60°$C, $-60°$C]
b	[$-60°$C, $-60°$C]
as	[-0.5, 0.5]
bs	[-0.5, 0.5]
rfb	[-0.5, 0.5]

limit should be set large enough to fulfill long duration flight as long as 3 km. Other state variables are applicable to most application where flight conditions are near-hover. Therefore, large fluctuations of velocity, angular rate are considered abnormal behaviors and should be taken care of.

A practical flight test where true emergency came across is taken as an example. We conduct the experiment on SheLion to test automatic hover capability. During the test, a "hold" command is issued at 271.2 second. The UAV is hovering until a malfunction of the control law happens, which leads to the heading angle out of control. As shown in Fig. 3.35, the yaw angular rate reaches more than 2 rad/s at time 338.5 second. Correspondingly, the emergency check function detects this abnormal value and immediately sets the current behavior to emergency as shown in Fig. 3.36 at exactly the same time at 338.5 second. Then in the emergency behavior, the inner-loop A6 is activated and sets all the control signals to the constant equilibrium values during hover behavior. With these control signals on the servo output channels, the helicopter can maintain its attitude to some extent and assist

ground pilot to switch to manual mode and land the helicopter safely. Please note that after 338.5 second, the emergency behavior is continuously assigned due to the fact that UAV is still behaving crazy with out-of-range state variables. But thanks to the graceful downgrading of the flight control performance, SheLion is saved by the ground pilot.

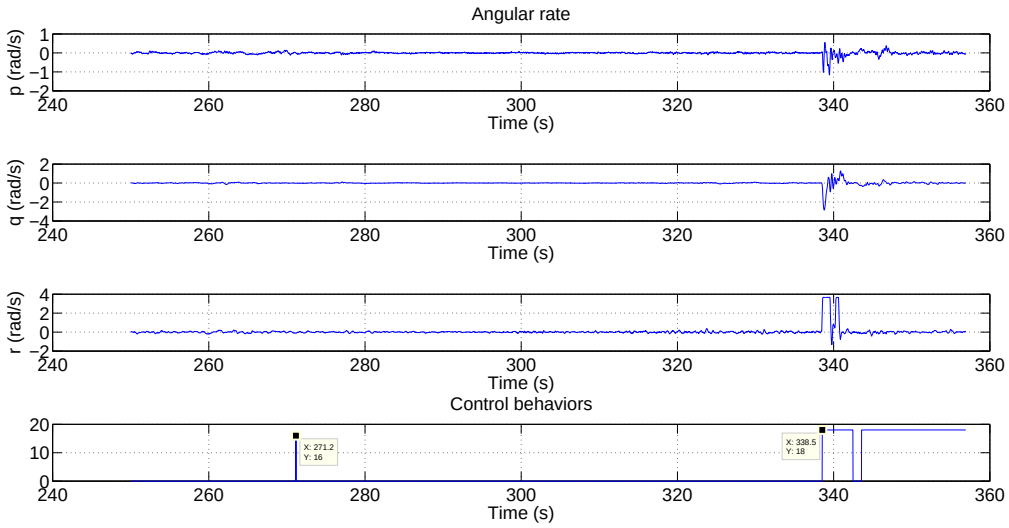


Figure 3.35: Measurement of angular rate during emergency

3.14 Conclusion

In this chapter, the onboard avionic software system is designed with data flow diagrams and UML diagrams in a top-down approach. Task identification and scheduling for onboard tasks are described. A behavior-based control structure for realizing scheduled flight task is also given. The software performance evaluations are also conducted and verified that the onboard software system can meet real-time requirements.

Further improvements of onboard systems include the mechanism for fault tolerance, incorporation of human experts, and etc. For fault tolerance, hardware redundancy needs

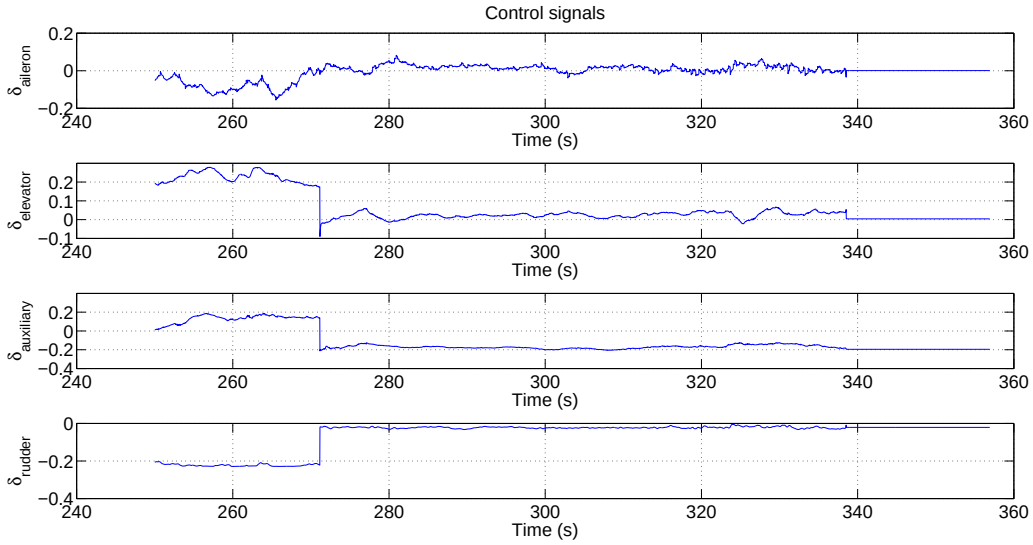


Figure 3.36: Control signals during emergency

to be implemented. In the perspective of software systems, another onboard avionic system can be mounted in parallel with the working one. In case of hardware failures, a detection and recovery mechanism should be able to activate the second avionic system and the flight task can be successfully continued. In addition, the experiences of human experts can assist in situations such as emergency, navigation and control in unknown environments. An application of semi-auto flight task is a good example, where the pilot can realize 3D movements based on the live images received on GCS.

Chapter 4

Software Platforms

4.1 Introduction

Given the real-time property of the onboard avionics system, a robust, reliable real-time operating system is thus critical to host the onboard software program such that the critical tasks with timing requirements can be satisfied and the peripheral hardware can be well supported. The task requirements with timing constraints are specified in Chapter 3. To achieve real-time property and more hardware components support, an avionic system with two main processors is developed. One processor is responsible for executing the tasks for automatic flight control. Another processor is dedicated to perform camera related processing including capturing and processing camera images, exchanging data with the control processor, transmitting live images to GCS. With two separated processors, the performance of flight control subsystem can eliminate the impact of malfunctions of the vision subsystem. The flight control subsystem is also convenient to be ported to other UAVs without too much modifications in either hardware or software. Also, the camera subsystem aims to develop as an independent module exclusively for image processing as well. On the other hand, with the collaboration of two subsystems, the avionic system can fully utilize various sensor data and processing power of two CPUs to fulfill complicated

applications such as vision-aided target tracking and navigation.

The software platform development of the avionics system involves processor selection, peripheral components connection design, and RTOS image customization based on a specific processor.

4.2 Processor Selection

The flight processors we focus are general-purpose unit, where an RTOS can be imported. Some key properties in selecting processors are listed below:

1. Rich peripheral hardware support: There should be enough commonly used hardware interfaces such as UART, USB.

2. Advanced RTOS support: The RTOS should be advanced enough to support various hardware drivers to facilitate data exchange. In addition, an RTOS should support efficient real-time task scheduling.

3. Compact dimension and low weight: As the processor performs tensive tasks during flight, the properites such as small size and low weight are important for agility of UAVs.

Based on the listed requirements, two kinds of processors are adopted for our onboard avionics development. One is the industry-standard PC104 computer stack [57], another is the small size yet powerful ARM architecture based OMAP3530 integrated on Gumstix Overo Fire [51]. The specifications of the two processors are listed in Table 4.1 and 4.2. Both processors are commercial-off-the-shelf products. The PC104 computer is widely deployed in the industry with many hardware support, while the Gumstix Overo Fire is widely adopted in embedded system with low power consumption. Due to the rapid development of embedded processors, the small compact yet high performance Gumstix is preferred in our avionic system. However, the onboard program development is designed to be cross-platform such that both X86 and ARM processors can be deployed over multiple different

UAV platforms with minor modifications. The PC104 computer stack and Gumstix Overo Fire are shown in Fig. 4.1 and Fig. 4.2.

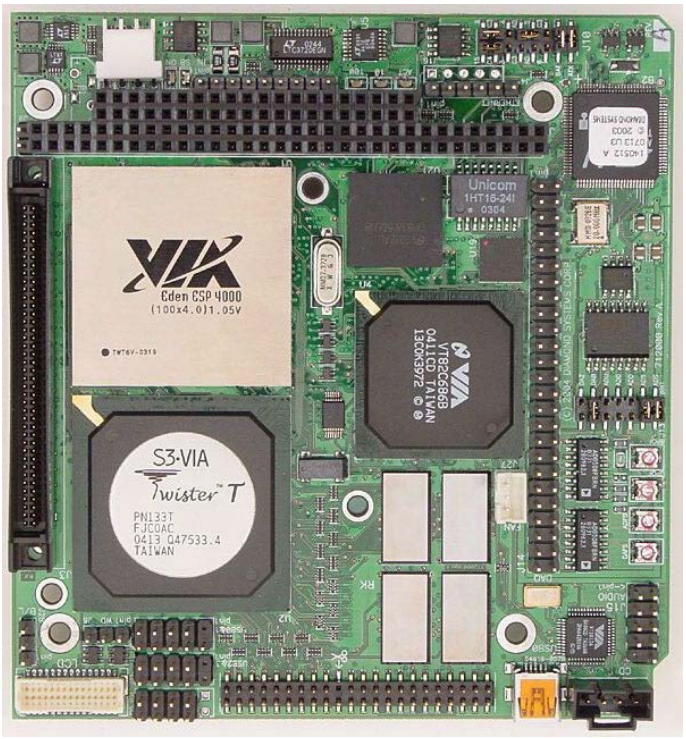


Figure 4.1: PC104 embedded single board computer: ATHENA

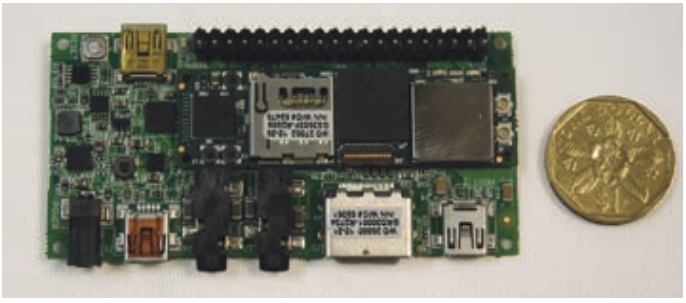


Figure 4.2: Gumstix Overo Fire processor

Table 4.1: Specifications of PC104 ATHENA

Processor	Pentium-3
CPU speed	400 MHz
Memory	128 MB
Serial ports	4 RS232
USB ports	4, version 1.1
Network	10/100 Mbps Ethernet
Power supply	+5VDC
Power consumption	10 watts
Dimensions	96mm x 90mm x 10mm
Weight	150 g

Table 4.2: Specifications of Gumstix Overo Fire with Summit expansion board

Processor	ARM Cortex-A8 CPU C64x+ digital signal processor (DSP) core
Clock	720 MHz
Memory	512MB RAM, 512MB Flash
Serial ports	3 UART
USB ports	1 host 2.0, 1 otg 2.0
Network	WiFi 802.11b/g
Power supply	+5VDC
Power consumption	1 Watt
Dimensions	80mm x 39mm x 8.2mm
Weight	22g

4.3 Avionic Components Integration

With the selected processors and pheriphral hardware components, the integation among them is needed to connect each hardware with the corresponding interface in the processor.

With the selected processors and sensors, the avionic system can be described in the diagram shown in Fig. 4.3. The avionics system is composed of two subsystems, control subsystem and vision subsystem. The control processor will process data from IMU plus GPS, laser scanner, send servo driving signal to servos and communicate with other UAVs and GCS. While the vision processor is to capture live images and perform image processing algorithms to provide high level navigation data for the control processor.

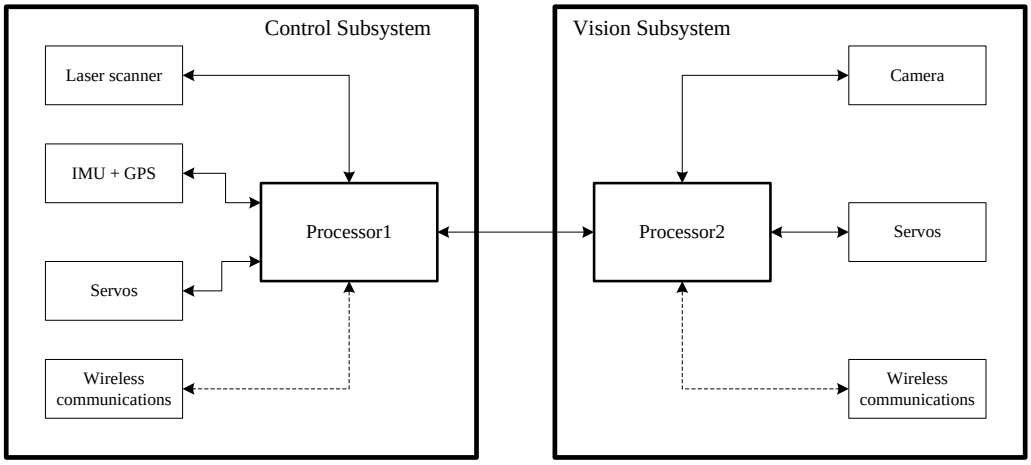


Figure 4.3: Onboard avionic system two CPU layout

As shown in the hardware layout diagram, this avionic system can realize different applications given corresponding configurations. For example, to realize pure automatic flight task, only the control processor and its peripheral components are used. On the other hand, if the ground target tracking mission is required, then both control CPU and vision CPU are activated cooperatively to fulfill the task.

4.3.1 PC104 Avionics Integration

As quite a lot of commercial hardware components support the industry-standard PC104, the integration work is relatively straightforward. Fig. 4.4 illustrates the selected hardware including UART based IMU, UART based servo controller, and UART based wireless modem. As listed in Table 4.1, there are four standard UARTs available on PC104. Therefore, three out of four UARTs are deployed to communicate with hardware. The last UART is used as the communication port with one UART of another PC104 in the vision subsystem. As such, the integrated flight control CPU can interact with all the hardware components. Fig. 4.5 shows the whole avionic system with two PC104 processors.

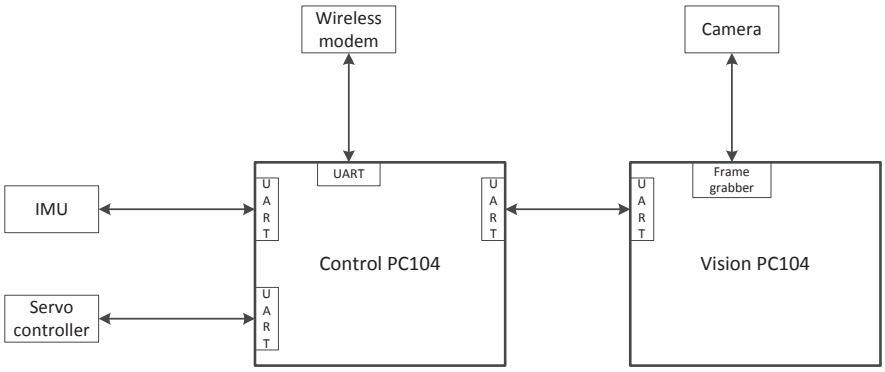


Figure 4.4: PC104 avionics integration

4.3.2 Gumstix Avionics Integration

For Gumstix avionic system, it needs to interact with the following hardware components in Table 4.6. It is clear that the Gumstix processor cannot afford to provide enough ports especially UART and USB to support peripheral sensors. Therefore, a USB hub is adopted to extend the USB ports up to five. Correspondingly, each USB port can be converted to UART port via the commercial-off-the-shelf usb2serial chip. The final integration of gumstix avionics system is demonstrated in Fig. 4.6. The corresponding customized printed circuit

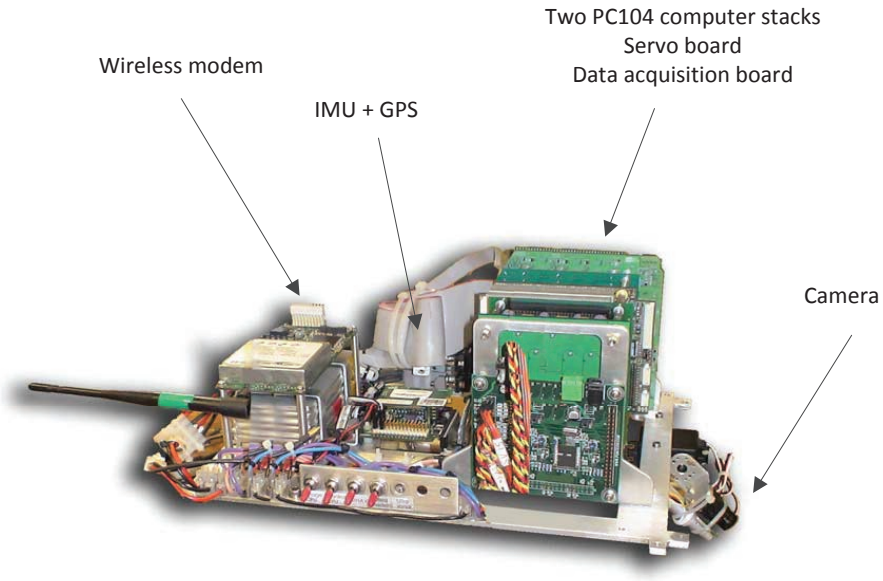


Figure 4.5: Avionic system with two PC104 processors

boards (PCBs) for control subsystem and vision subsytem are shown in Fig. 4.7. We name them LionHub and VisionHub, respectively.

4.4 Operating System Customization

As explained in Fig. 3.1, the under layers below the user applications should be customized to suit for various requirements such as real-time, rich peripheral interfaces. In this section, the OS customization is focused on Gumstix based avionic system. As the embedded system consists of two subsystem, the operating system needs to be customized for the two processors, respectively. As described in Chapter 3, the flight control processor plays a vital role in the whole flight task where real-time requirements must be met within the deadline in each execution loop, an industry-standard operating system, QNX Neutrino, is adopted. On the other hand, the vision processor needs various high class cameras to achieve good processing results, a Linux distribution with minimal file systems and rich camera driver

Table 4.3: Gumsix avionic system components

Processor	Hardware component	Interface
Control gumstix	IMU	UART
	Servo controller	UART
	Wireless modem	UART
	DAQ board	UART
	Laser scanner	USB
	Vision gumstix	UART
Vision gumstix	Camera	USB
	3G modem	UART
	Control gumstix	UART

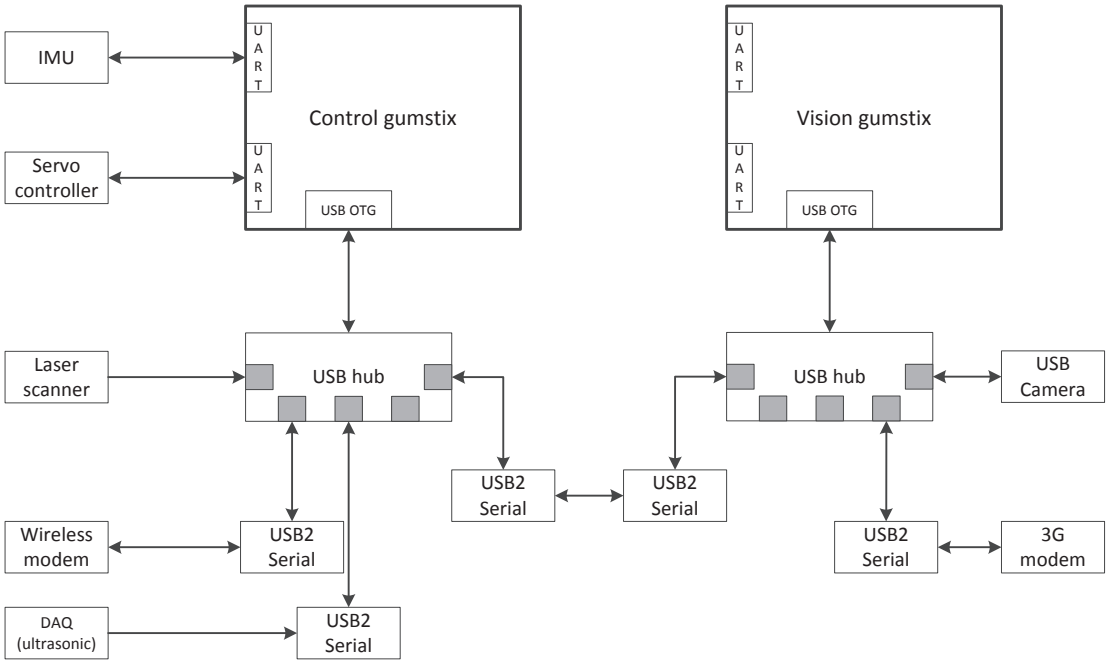


Figure 4.6: Gumstix avionics integration

support is deployed on it. The following sections will provide details of OS customization on each processor.

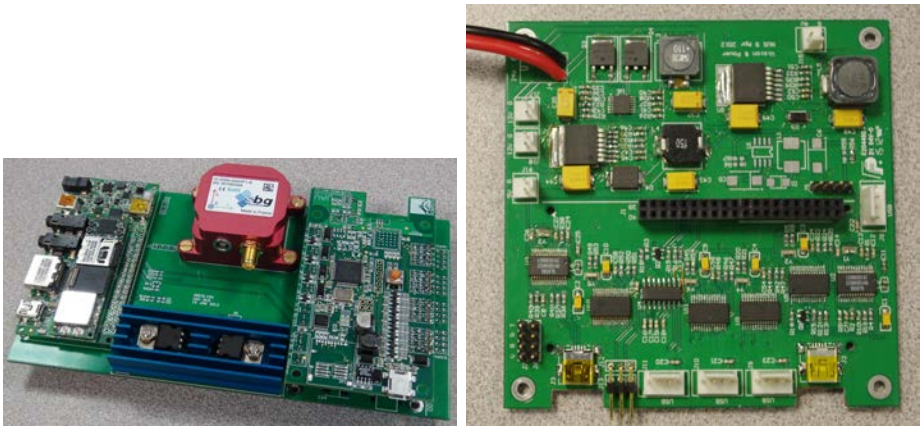


Figure 4.7: Avionic system with two Gumstix processors

4.4.1 QNX Neutrino

QNX Neutriono is developed by QNX Software Systems, Canada. QNX is a true microkernel operating system. Under QNX kernel, all device drivers, user applications, file system, networking run outside the kernel, with the most fundamental OS primitives executed within the kernel, which is a unique feature among RTOS. Besides, the QNX Neutriono also possess other valuable properties which make it a suitable solution for mission-critical application designs such as avionics flight systems.

For real-time critical applications, such as autopilot system, the operating system on which the user application is running greatly affects the critical performance of the real-time applications. As such, the development of avionic software system is dominantly carried out in a real-time operating system (RTOS) environment, which can effectively guarantee the final system executed in a deterministic behavior, based on certain scheduling, intertask communications, resource sharing, interrupt handling, and memory allocation algorithms [12]. Currently, three most popular real-time operating systems adopted in the UAV development are the QNX Neutrino [63], VxWorks [67], and RTLinux [64].

The QNX company provides the source code with a board support package (BSP) [62] for popular processors in the market. The Gumstix overo is based on OMAP3530 processor

and thus its BSP source code is necessary to build the system image. The QNX image is developed in the QNX Momentics IDE 4.7.0. The BSP source code can be categorized into the following sets: IPL (initial program loader), startup modules, buildfile, board-specific device drivers, system managers, utilities and etc. The IPL performs enough hardware initialization to load and start the OS. The startup module further initializes the hardware and provides board dependent code for QNX microkernel. The buildfile is a build script for loading libraries and drivers that will run on the Gumstix. The drivers included in the source code are actually the hardware which the Gumstix can support. System manager is for customizing user drivers, while utilities are commonly used commands in QNX. Given the avionics integration described in Table 4.6, the startup module needs to modify to activate second UART which is not available in the provided BSP. Also, considering the WiFi capability on the Gumstix, the WiFi driver needs to be modified and loaded during startup.

Activating the Second UART

As the pins of UART2 are routed for Bluetooth control by default in the motherboard of the gumstix, pin configuration needs to be modified during startup. The source code is located at: bsp-TI-omap3530-src/src/hardware/startup/boards/ omap3530/init_pinmux.c. The above modifications are listed below based on the datasheet of omap3530 [36]:

```
        // change bit mask for UART2 pins
        out32(CONTROL_PADCONF_UART2_CTS_RTS, MUXMODE1_MODE4 | MUXMODE0_MODE4);
        out32(CONTROL_PADCONF_UART2_TX_RX, MUXMODE1_MODE0 | MUXMODE0_MODE0 | INPUTENABLE1);

        // Pins for UART2 (BT Control)
//      out32(CONTROL_PADCONF_MCBSP3_DX_DR, MUXMODE1_MODE1 | INPUTENABLE0 |
PULLTYPE0_UP | PULLUDENABLE0 | MUXMODE0_MODE1);
//      out32(CONTROL_PADCONF_MCBSP3_CLKX_FSX, INPUTENABLE1 | PULLUDENABLE1 |
MUXMODE1_MODE1 | MUXMODE0_MODE1);
```

Add WiFi Support

The hardware built-in WiFi capability is a valuable feature for embedded avionic system which can realize wireless online debugging, file transfer and system administration work. In addition, the peer-to-peer (P2P) architecture of WiFi network is also suitable for the multi-UAV cooperative tasks.

The network is configured with WPA method [68] to assign the mode as Ad-Hoc, the network name as "gumstix100", IP address as "192.168.100.3" and password as "1234567890".

```
        io-pkt-v4 -d /proc/boot/devnp-mv8686-mmc2.so dir=/root/io-pkt,verbose=1,poll=0 -p tcpip
mclbytes=4096,stacksize=65000 random
        waitfor /dev/mv0 5
        wpa_supplicant -Dwext -imv0 -c /etc/wpa_supplicant.conf &
        sleep 1
        ifconfig mv0 192.168.100.3

/etc/wpa_supplicant.conf={
ctrl_interface=/fsram/tmpfs
eapol_version=1
ap_scan=2
network={
        ssid="gumstix100"
        wep_key0=1234567890
        key_mgmt=NONE
        mode=1
}
}
```

The development host running Windows 7 also needs to connect the WiFi module with the network "gumstix100", manual config the IP address to be in the same wireless local area network (WLAN) with the Gumstix WiFi in a peer-to-peer architecture. To add another Gumstix avionics to this WLAN, a different IP address such as "192.168.100.4" should be assigned. Thus, a small P2P network is established between two Gumstix avionics , which one can communicate with the other.

Mounting QNX OS Image

The whole OS image is built read only where no user files can be executed or written on the file system. In our application, the executable application should be able to store on the file system and to execute each time the board starts. Also, the log data of each flight should be stored as well. A 2GB micro SD card is used to store the OS image, flight application and log data. The script in the buildfile to mount a micro SD card at "/fs/sd" and set read, write and execute permission is:

```
devb-mmcsd-beagle cam quiet blk cache=2m,automount=hd1t6:/fs/sd mmcsd noac12 dos exe=all
```

4.4.2 Embedded Linux

The Linux OS image customization is done for the vision Gumstix processor. Considering the hardware components needed in Table 4.6, the Gumstix needs to support USB cameras from various vendors. For image processing, the OpenCV library is needed to use various functions for real-time computer vision applications.

Image Build Environment

We adopt a powerful community solution OpenEmbedded[54] to build the system including the necessary components such as bootloader, Linux kernel, root file system consisting of hardware drivers and libraries, cross-compilation toolchain. The OpenEmbedded adopts the BitBake[49] as the cross-compilation for compiling thousands of open source packages available in the repository of OpenEmbedded. Fig. 4.8 illustrates the procedures during the system build with BitBake. The BitBake will perform the following preparations:

1. Parse all the recipes from OpenEmbedded repository;

2. Fetch all the source code to be built in the target system;

3. Apply patches to all the source code;

4. Apply build configurations and starts building;

5. Download the generated OS image including bootloader, Linux kernel and root file system to the micro SD card.

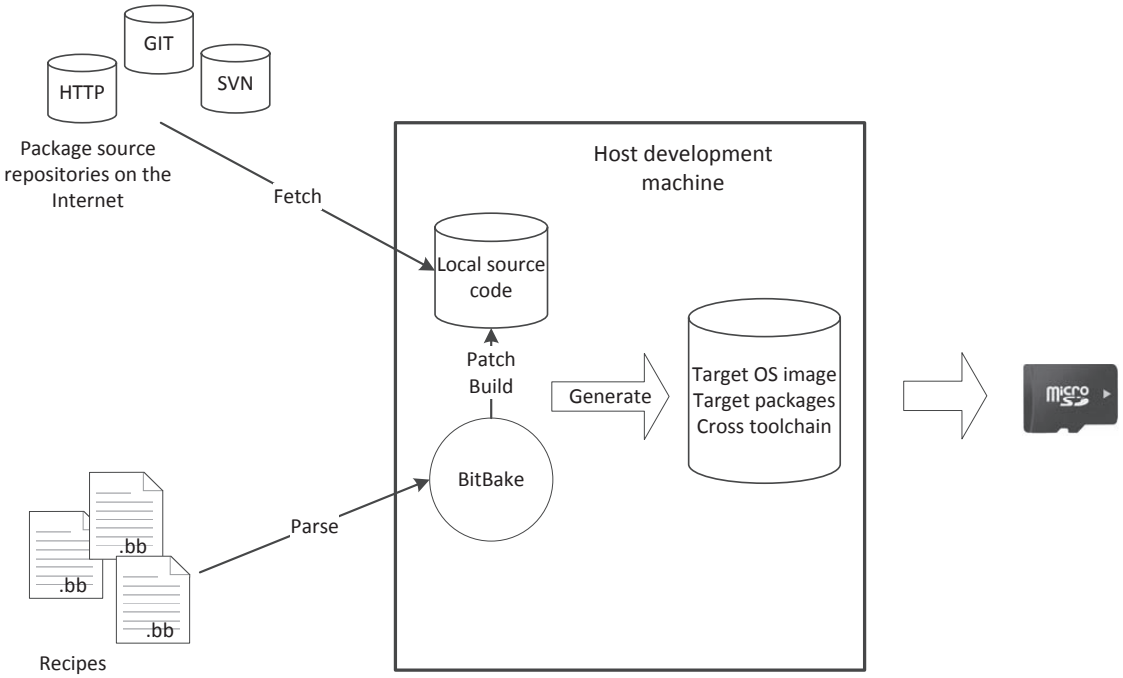


Figure 4.8: OpenEmbedded system build procedures

Image Customization

As discussed earlier, the OpenCV library is heavily involved in the vision Gumstix processing, it is reasonable and convenient to incorporate OpenCV libraries directly in the root file system. Among the target recipes, the image recipe which builds a set of packages and combines them into an image such as a console image is adopted as a basis image. Based on this base console image, a customized image recipe is created with OpenCV included. The customized image recipe "omap3-nusopencv-image.bb" is modified as below:

```
require omap3-console-image.bb
```

```
IMAGE_INSTALL += " \
  opencv \
  opencv-dev \
"
export IMAGE_BASENAME = "omap3-nusopencv-image"
```

Build User Applications

It is also necessary to build self developed applications into the image file system. For example, vision Gumstix requires the camera application to execute automatically after logging into the system. To achieve this, a separate user recipe needs to be created and then built and packaged with BitBake. The steps are summarized as below:

1. Create the user application. In this case, an automatic login of the Linux OS after power-up is written as shown in Fig. 4.9.

autologin.c

```
#include <unistd.h>
#include <stdio.h>

int main()
{
        int nrv = 0;
        FILE* fptr = 0;
        char user[64];

        // clear buffer
        memset(user,'\0',64);

        // open autologin profile file
        fptr = fopen("/usr/autologin.profile\0","r\0");

        // make sure the file exists and was opened
        if (fptr != 0)
        {
                // the return value from fscanf will be 1
if the autologin profile name is read correctly
                nrv = fscanf(fptr,"%s\0",user);
        }

        // only autologin if the profile name was read
successfully,
        // otherwise show the regular login prompt
        if (nrv > 0)
                nrv = execlp("login\0","login\0","-f
\0",user,0);
        else
                nrv = execlp("login\0","login
```

Ln 19, Col 38 INS

Figure 4.9: User application program

2. Create a user recipe for BitBake to build within the OpenEmbedded framework [54]. The script is shown in Fig. 4.10. It specifies the source file to build, autologin.c in this case, and corresponding flags for compiler BitBake.

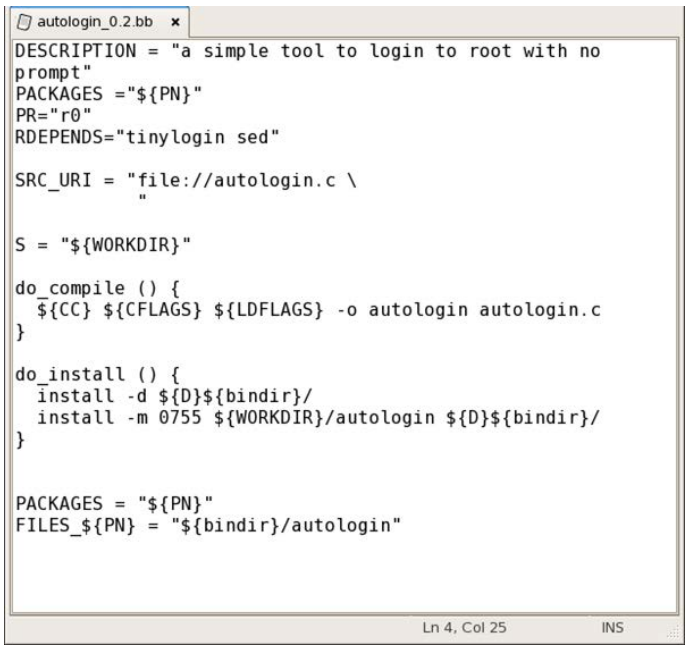


Figure 4.10: A recipe for the automatic login program

3. Once the build is successful, a package for Gumstix is generated in the OpenEmbedded tree and is shown as in Fig. 4.11. Then the user can manually install this package and automatic login function can be realized when Gumstix is started.

4.5 Conclusion

In this chapter, the software platform development of onboard avionic system is presented. The RTOS image customizations of both QNX and Linux make the two-processor solution more powerful with integration of more peripheral sensors such as laser and camera. With two powerful processing units and rich environment information, more challenging missions can be achieved. Considering the future computing requirements, multi-processor system

Figure 4.11: Autologin package generated in the OpenEmbedded

would be the solution. As such, it would be necessary to consider adding another processor to the onboard avionic system. Given the three-processor layout, the computation loads can be categorized into three tasks: sensor fusion, vision processing and control law calculation. The sensor fusion processor must have enough hardware interfaces to retrieve various peripheral sensors including the vision processing output in a synchronized approach. With the synchronized data, the optimized control performance can be easily implemented. The communication and synchronization for multi-processor can be designed as suggested in [24].

Chapter 5

Ground Control Systems

5.1 Introduction

The ground control system is also an integral component in the whole UAV systems as illustrated in Fig. 2.2, which acts an interface between GCS operator and unmanned vehicles. Current military GCS examples include Predator system[60] and Global Hawk[70], both of which can perform highly complicated missions such as human-in-the-loop remote control, overwrite UAV flight course, target location and attacking and etc. The GCS of Predator and Global Hawk are shown in Fig. 5.1. From the GCS external layout, we can see that although it possesses powerful and comprehensive processing capability, it requires quite a few people to operate and maintain. Furthermore, a GCS with such standards would cost millions of dollars. Considering our research focus is on the small-scaled UAVs with civilian applications, a low-cost and user-friendly GCS should satisfy our needs.

There are world wide universities and research institutes who developed their own ground control systems, which is light-weight, fully functional and can be a good reference for our own development. The Pixhawk [58] UAV team develops the open source QGround [61] with a widely adopted communication protocol MAVLink. Another widely adopted GCS is the one built by DIYDrones [48] which user can modify the onboard software system to

Figure 5.1: GCS examples - Predator and Global Hawk

be customized for a particular UAV platform. From the above popular GCS design, we can summarize the following functions or features that a GCS should have: 1) UAV status feedback and display; 2) task management or mission planning where user can control the behaviors of UAVs; 3) hardware-in-the-loop simulation to verify the design of UAV systems.

5.2 Software Modeling

The requirements of GCS are mainly concentrated on the user interactions with GCS. Specific operations such as selecting a view of a certain type of data, zooming in and zooming out, reading out the values, and etc. are carried out by GCS operator. In order to fully describe the interaction details, the *Use Case* diagram of UML [17] is adopted to represent the relationships between the user and GCS. A use case can capture the primary functions of the system in a broad view without bothering technical details. It consists of actors, actions and associations. An actor is an object outside the scope of system that have actions towards the system. The association is to link the actors and actions happened within the GCS system.

Fig. 5.2 represents the activities between GCS operator and UAVs. There are five major use cases including: Setup connections, Information monitor, Task management, Playback and Provide flight status. A detailed decomposition of *Information monitoring* is also provided in Fig. 5.3, where specific user operations on a certain view are illustrated. The

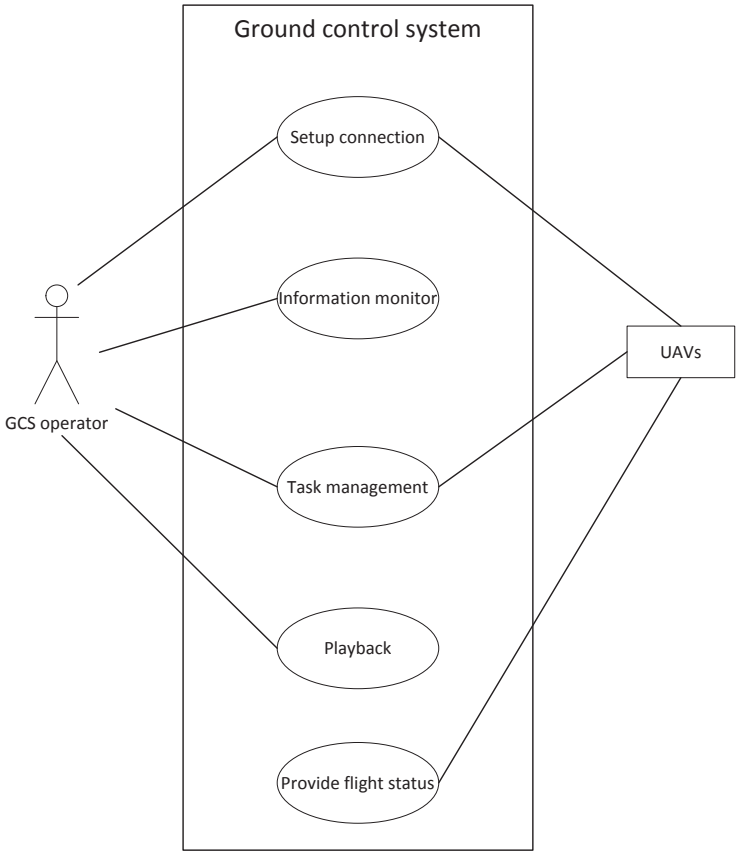


Figure 5.2: Use case diagram of ground control system

relationships among use cases are described here. When a use case provides an additional capability for another use case, a notation of "extends" is applied between the two use cases.

The GCS first sets up connections with UAVs, then starts listening on the communication ports from each UAV. The connections refer to the hardware initialization including serial port, TCP/IP network configurations. On receiving flight status data, the GCS operator can view them in several approaches as shown in Fig. 5.3. The details of user activities in *Information monitor* are listed in Table 5.1.

Given different requirements to facilitate GCS operator monitoring, the *Information*

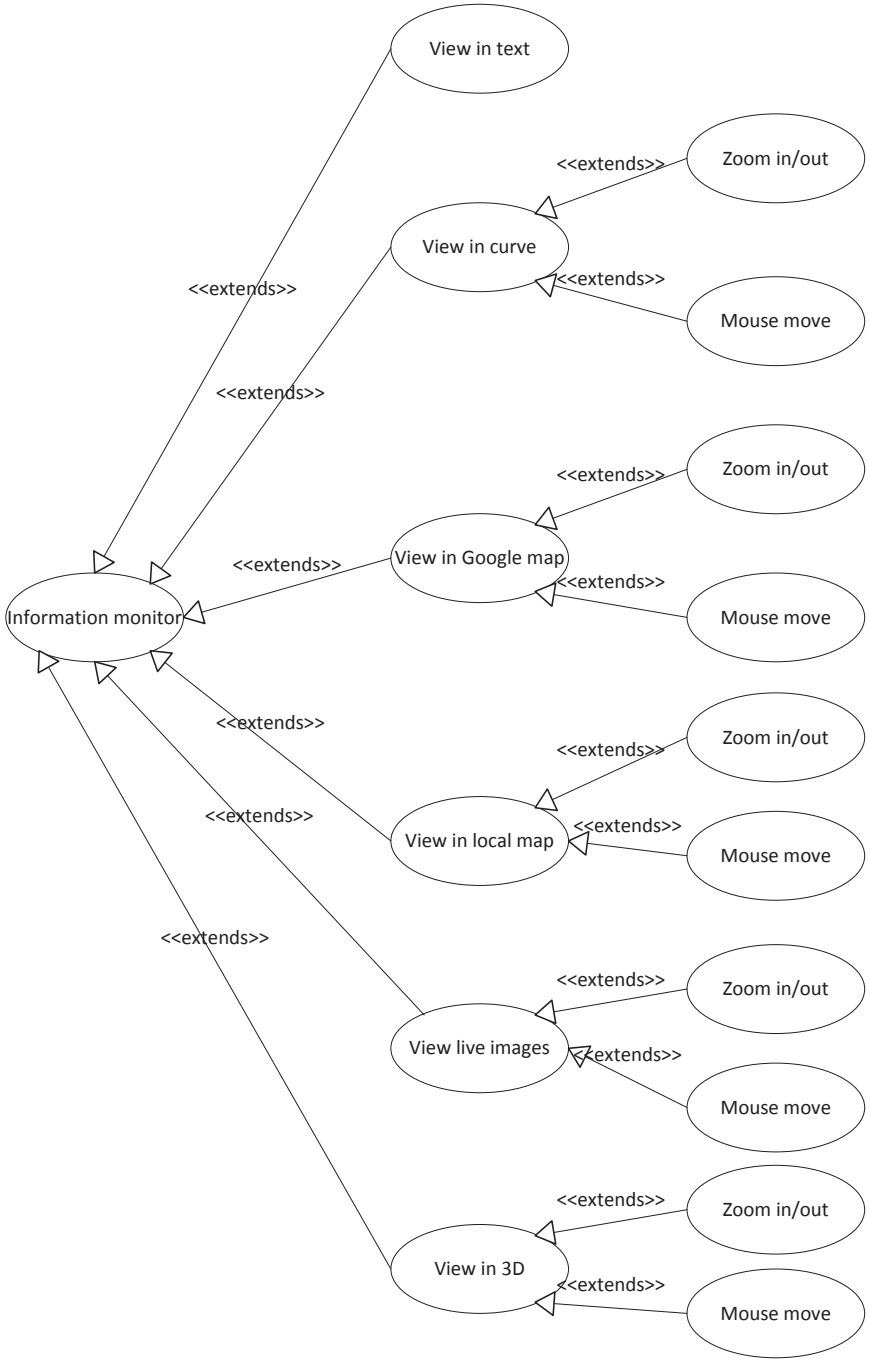


Figure 5.3: Use case diagram of information monitoring

Table 5.1: GCS operator activities

Activities	Task	Description
View in text	Select and double click	Double click a certain variable in the view list, then the curve view of this variable is displayed
View in curve	Zoom in	Scroll up the mouse to see data in smaller scales
	Zoom out	Scroll down the mouse to see data in bigger scales
	Read data	Once mouse is moved over the area of the curve, the readings of the mouse point is displayed
View in Google map	Upload waypoints	Select certain UAVs on Google map, then press Enter to upload to UAVs
View in local map	Zoom in	Scroll up the mouse to see data in smaller scales
	Zoom out	Scroll down the mouse to see data in bigger scales
View live images	Zoom in	Scroll up the mouse to enlarge images
	Zoom out	Scroll down the mouse to reduce images
	Specify ROI	Click on the image, the combination of each points are transmitted back to UAVs
View in 3D	Change perspective	Move mouse to change perspective

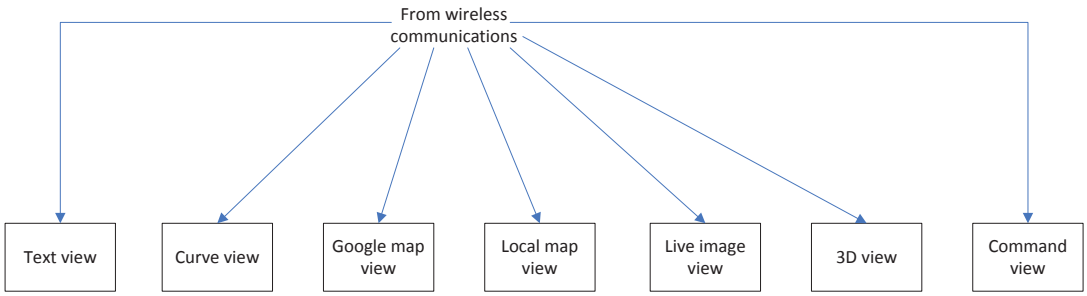


Figure 5.4: Information monitoring components

monitoring in Fig. 2.2 can be further decomposed into the following submodules as listed in Fig. 5.4. Another module in Fig. 2.2 is the *Task management*, in which the GCS operator issues a command and UAV acknowledgment message of this command will be displayed.

5.3 Software Architecture

The software on GCS is realized with MFC (Microsoft Foundation Class) in a laptop with the Windows 7 Professional operating system. The overall architecture is realized via the SDI (Single Document Interface) approach as shown Fig. 5.5. The MFC application wizard provided by Microsoft builds the application with a document class and multiple view classes separately. The document class manages the data store, data processing and coordinates the data display in multiple views. While the view class is responsible for data displaying and interactions with the user such as selecting, clicking on the views. In addition, each view represents a separate view of the data, while the command data processing routines can just reside in one document class. The document class manages the updating of views when the document data updates. As such, most of the development is to implement the document class and multiple view classes. The system commands and messages can be embedded into the document and view class and fulfill the overall GCS specifications.

The GCS is composed of background tasks and foreground tasks. The background layer has mainly two tasks, receiving flight status from and sending commands to multiple UAVs,

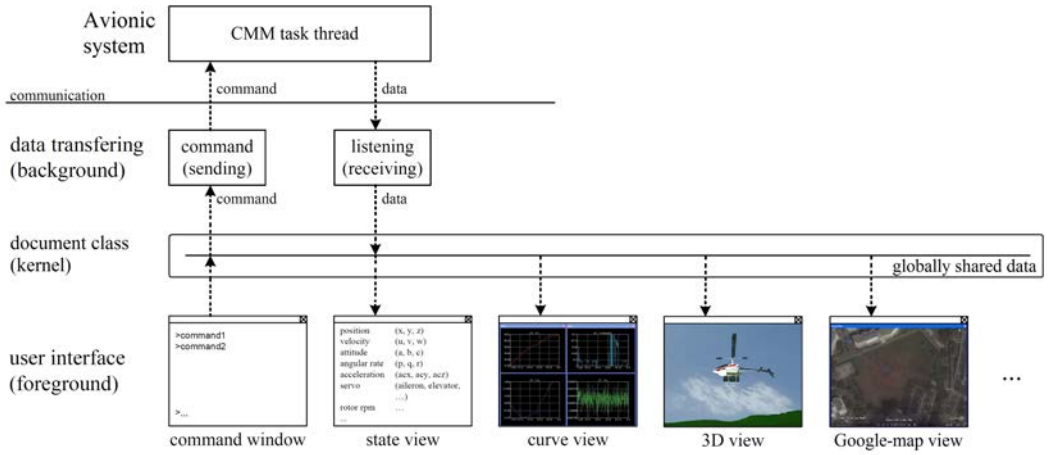


Figure 5.5: Architecture of ground control system [4]

both of which interact with the UAV onboard CMM task. The receiving thread accepts all the data from the fleet of UAVs, and identifies each status data via the telegraph packet header. Consequently, the corresponding multiple display is executed, and the cooperative waypoints of the paths are demonstrated. Similarly, the upload link can broadcast the commands to all UAVs, or alternatively send commands to a specific UAV, both via the sending task. The global status data from UAVs are dynamically updated from the background layer. The foreground task composes of *Information monitoring* and *Task management*, where the *Information monitoring* module consists of various user-friendly views. A document class implementation in MFC is deployed to realize the communication between the background tasks and foreground tasks. The document class performs the flight data receiving, data processing, command interpreting and packaging, and etc.

The execution scheme for realizing the GCS software framework can be described in the following pseudo-code:

document class

init

create communication threads

(pass pointer to this document to threads as parameter)

create views

(pass pointer to this document to these views)

create timer

ontimer

loop {

get pointer to next view

if the pointer is null(end), exit loop

call the updating function (onupdate) of that view

}

view class

onupdate

get the pointer pointing to the document

get new data in the document

draw view according to the new data

communication thread 1 (data-receiving)

loop {

read serial port for communications

if new data received {

storage new data in the document

}

}

communication thread 2 (data-transmitting)

loop {

```
  look up to the document if new command captured
  if there is new command {
    translate command and parameter in telegraph
    write telegraph to communication serial port
  }
}
```

In this program, we should pay special attention to the following three key points:

1. The *init* function is executed to initialize the document class and to create the communication sending and receiving threads, various data views and updating timer. When these threads and views are successfully created, a pointer of the document is assigned to them to establish the necessary links. The document class itself stores an array of pointers.

2. The timer is created to send updating signals periodically. The timer message is handled by the member function *ontimer*. In every cycle, when the timer message is captured, the ontimer function searches all views linked to the document class and calls their updating functions (*onupdate*) sequentially.

3. A universal method is used to define all the view functions in the view class. The *onupdate* member function is called every time when the shared data storage is updated by new inflight data packages. In *onupdate*, it is programmed to first obtain the pointer, which points to the document class, and then access the associated stored content, before a drawing function is called to display new data.

4. The kernel layer is to decode the user command to a format that can be recognized by the avionic software system. Once a new user command has been received, the kernel layer calls a function to obtain the string and translates it in an internal representation carrying command code and parameters. The translated code and parameters are then stored in the globally shared data storage.

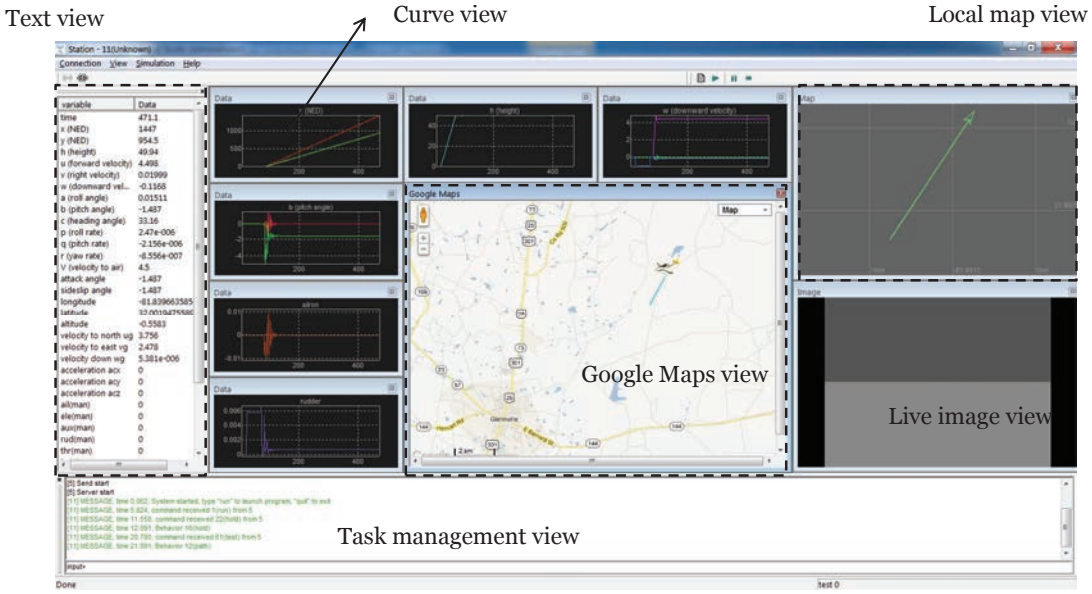


Figure 5.6: Screenshot of GCS layout

5.4 Information Monitoring

In the *Information monitoring* module, several submodules are developed to realize multiple user-friendly views for the ground user. Besides, this module can receive the commands regulated by the *Task management* module which is responsible for interpreting user input commands to executable tasks for individual UAV members. Specifically, five kinds of views are developed, *Map view*, *Curve view*, *Text view*, *Live image view* and *3D view*, which are all shown in Fig. 5.6. All views can be dynamically loaded once GCS starts running can configured by user.

The design of *Information monitoring* involves the display windows and specific operations on the selected windows. The display windows render different kinds of views of data to the operator, such as text and curve. The window rendering design involves the window layout design, and the data associated with the corresponding window. On the other hand, the operations associated with the selected windows are realized via the system built-in window messages and corresponding handlers. The user can also create self-designed messages

and handlers [31] associated with them.

5.4.1 Text View

The text view is to display the values of each data variable of the flight status data in the numerical format once the new data is received. The text view is necessary that it can display all kinds of UAV data including sensor data, manual servo data, automatic control signals and etc. The text view list gives the GCS operator a complete view of the flight data variables such as position, velocity and acceleration in three dimensions, Euler angles and angular rates and etc. as shown in Fig. 5.6. By double clicking on the data variable, the corresponding curve view with historical data will be created and displayed.

The *Text view* is implemented via the *CListCtrl* view provided in MFC. The *CListCtrl* view is encapsulated to manage a collection of items each consisting of an icon and a label. By adopting the structure LV_COLUMN, a two-list view is created by inserting all the variable names and corresponding data values. Once the one second timer message arrives, the second column representing the data value is updated for each variable.

5.4.2 Curve View

Given the time varying nature of the updated status data from UAVs, the flight data is preferred to display with the varying time. As such, a dynamic view with time as x-axis and data values as y-axis is proposed. The *Curve view* is activated by user double clicking on a certain variable on the *Text view*. However, due to the screen size limitation, only a few numbers of the curve view windows can be displayed. With the curve view, GCS operator can explore the behaviors of the UAVs given the changes of the curve such as transition from hover to forward flight in the velocity in body frame.

5.4.3 Map View

The map view is a straightforward way of displaying the flight trajectories of the UAV team. With the geological information, the whole mission plan of the UAV team is easy to monitor and manage. Considering outdoor and indoor condition, two kinds of map view are developed. One is the Google Maps view, where the GPS information consisting of latitude and longitude is updated in real-time according to the flight status data received in GCS. The other one is the local map view, which is suitable to the applications where GPS information is not available such as indoor environment. As such, a map indicating relative distance information will give the GCS operator an idea where the UAV is located given the starting point.

Google Maps View

With the utility provided by Google Maps, users can apply the APIs with their GPS data to dynamically display the trajectories by connecting the waypoints with the drawing methods provided in APIs. To use the Google Maps APIs, another separate JavaScript based application is developed to embed the APIs in the html websites. In this case, the MFC based application is combined with the Javascript based application to realize real-time waypoints display on the Google Maps. A database server application is also developed to realize the data write and retrieve operations to manage the data communication between the two GCS applications. The overall architecture is illustrated in Fig. 5.7.

The snapshot of the two GPS applications in the simulation mode is shown in Fig. 5.8 and Fig. 5.9.

Local Map View

As mentioned above, the local map view is generated by calculating the relative 2D distance given the starting point. The UAV is represented with a triangle. The direction of the triangle also reflects the heading angle update during the flight.

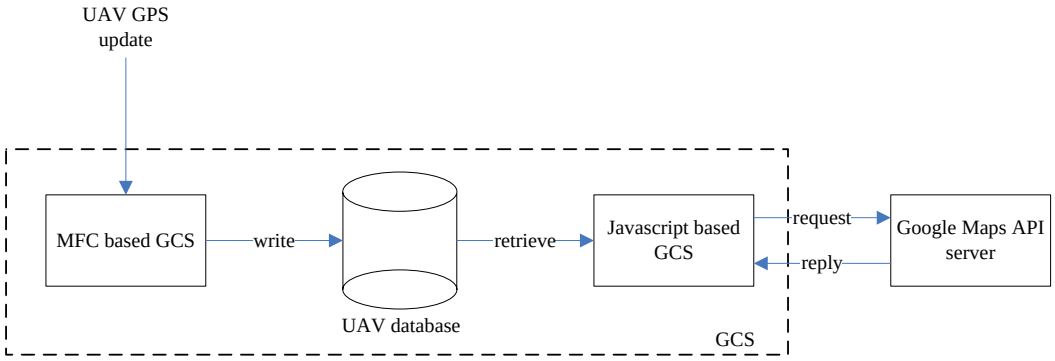


Figure 5.7: The architecture of GCS with dynamic map view

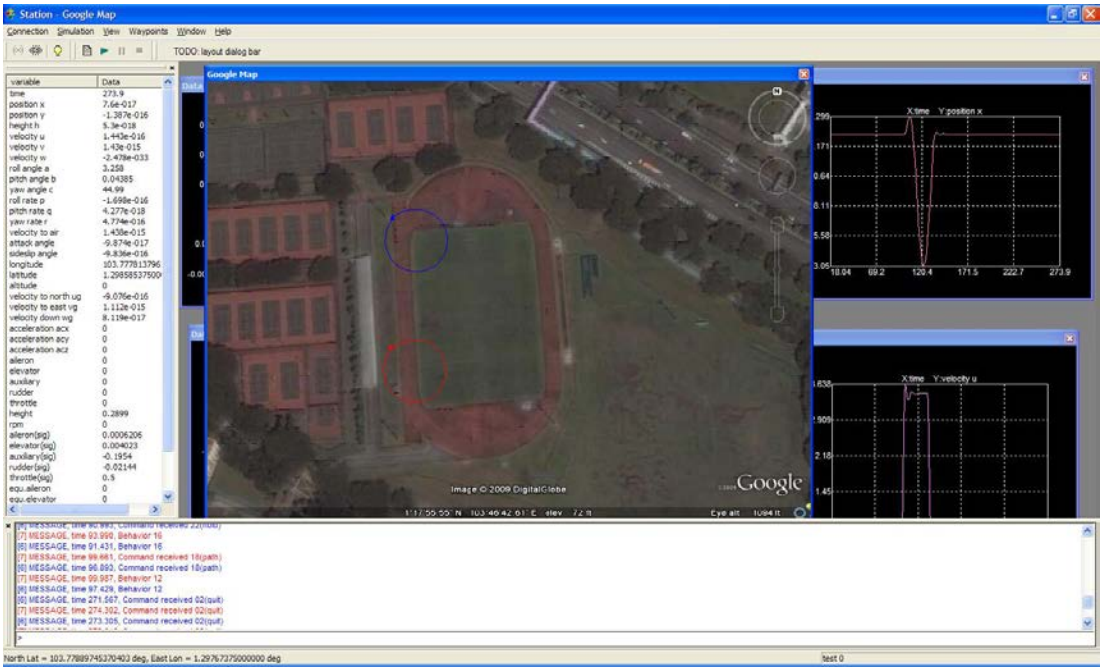


Figure 5.8: Screenshot of GCS with Google Map view

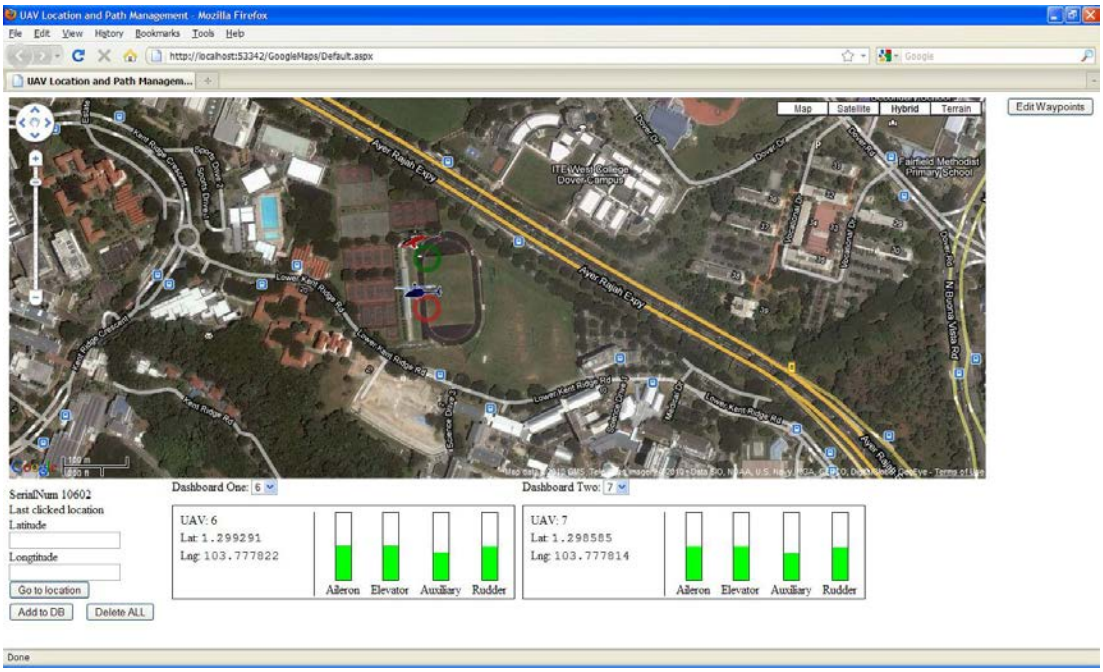


Figure 5.9: Screenshot of Javascript based application

5.4.4 Live Image View

The live image view is to relay the images taken by the onboard camera to GCS operator. With the live images, the operator can visualize where exactly the current camera is pointing at. It is especially instrumental when the UAV is performing vision based tasks such as target tracking. Besides monitoring purpose, the GCS operator can also upload the region of interest (ROI) in the picture to UAV vision subsystem to track this ROI. The operator first selects the current picture by freezing the frame update, then clicks on the picture with continuous four points which finally forms a four-point polygon. The coordinates of these four points in the frame are uploaded once the operator issues the command. Fig. 5.10 shows the live image view with user specified ROI within it.

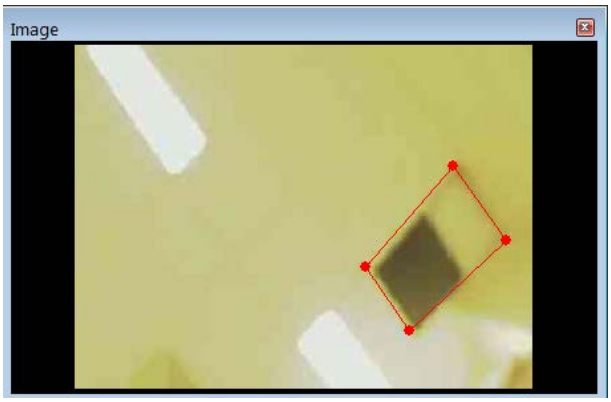


Figure 5.10: Live image view with specified ROI

5.4.5 3D View

The UAV 3D view development is another important module in the user interface design [14]. With the realistic 3D view rendering, the movements of the UAV can be monitored in a more mimic and convenient way for GCS operator. In addition, the 3D view can be used for remote monitor and control if the real helicopter is beyond pilot visible range. The development of 3D view involves three phases which is shown in Fig. 5.11. The first step is to establish the model the UAV helicopter platform constructed within our group. In addition, the flight environments such as sky and terrain are also modeled in the 3ds Max software. Next, the model objects are loaded by OpenGL drawing for movement transformations like rotations and translations to render the realistic response of the helicopter during the flight. The OpenGL also provides the functions such as changing the viewing perspective and painting textures to the 3D objects. The last stage is to realize the dynamic 3D view update based on the coming status data.

Model Development

The model development consists of the modeling of UAV helicopter, the terrain and the sky. In computer graphics, the 3D object model is described by a series of polygons and vertices.

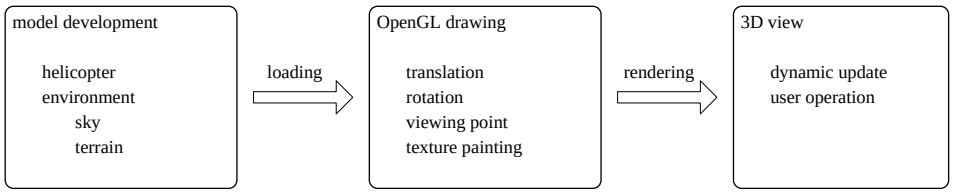


Figure 5.11: 3D view development

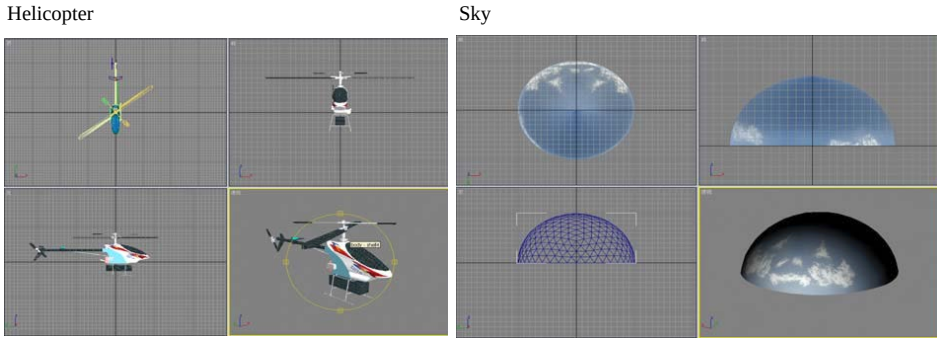


Figure 5.12: Model development in 3ds Max

Each vertex is described by a triple of coordinate numbers representing its position. While each polygon is described by a set of vertices as the boundary points of the objects. As such, the simulation of 3D object movement is by manipulating the transformations of the vertices and polygons in the kinematic way. The models developed in 3ds Max including helicopter and virtual environments are demonstrated in Fig. 5.12. The models are stored in .3ds files for OpenGL to load and manipulate.

OpenGL Drawing

OpenGL (Open Graphical Library) is an open source software designed for drawing 3D objects. The library provides the mathematical manipulation functions to realize translation, rotation of vertices, polygons of the objects [43]. Thus, the dynamic update of a object is accomplished by performing the manipulations on all the vertices and polygons of this

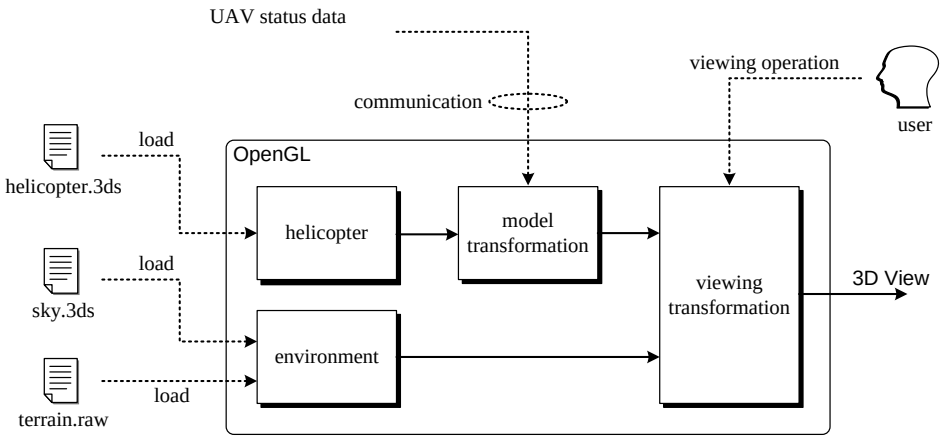


Figure 5.13: Draw in OpenGL

object. The overall structure to illustrate the drawing function with OpenGL is shown in Fig. 5.13. First, the model description files including helicopter.3ds, sky.3ds and terrain.3ds are loaded into OpenGL drawing module with the specific data format. Based on the received UAV status such as attitude and position, the rotation and translation operations are performed on the models to create a virtual realistic visual effect. The helicopter model consisting of helicopter body, main blade and tail blade is first manipulated by the model transformation, then the viewing transformation is applied to the model with the viewing perspective. Thus, the final 3D rendering of a helicopter in a virtual environment is realized.

3D View Update

The 3D view is lastly encapsulated in a class in C++ program. It is linked to the globally shared data, in which the inflight data of the UAV helicopter are stored. To dynamically simulate the motion of the helicopter, the view is updated with a rate of 10 Hz. In every cycle, latest values of the helicopter state are read from the global data pool and the content of the 3D view is repainted accordingly. The periodical update is scheduled in the program by a timer, which is created in the initialization stage of the 3D view. In the end of each cycle, a timer message will be sent to the view to request an update, which proceeds as in

the following code.

```
CTDView::OnUpdate()
{
  double pos[3] = { _data[1], _data[2], _data[3] };    //Local-NED position
  double att[3] = { _data[7], _data[8], _data[9] };    //Euler angles
  angle += rate*period/1000;    //incremental rotor angle
  SetPositionAttitudeRotor(pos, att, angle);    //calculate transformation
  GLDraw();    //redraw
}
```

The *OnUpdate* function is a member of the 3D view class *CTDView*, which is called whenever a timer message is captured. The front two lines get the local-NED position and Euler angles of the unmanned system from the global data pool, which is updated from time to time as the flight test progresses. The angle of the rotor is added by an incremental value every period to emulate the rotating effect of the UAV main rotor. Then the member function *SetPosiitonAttitudeRotor* is called to assign these states to the 3D view. This function is to calculate transformation matrices and apply them to the 3D objects. The *GLDraw* member function is finally called to redraw the content of the 3D view.

Finally, the appearance of the 3D view is illustrated in Fig. 5.14. Pictures are captured when SheLion is performing a full envelope automatic flight test. Some virtual views obtained from different viewing points are listed in the left and right columns.

5.5 Task Management

The task management module performs from basic automatic commands to individual UAV such as hover, forward flight to the high level mission planning for the UAV team given a

Figure 5.14: The 3D view of the helicopter in GCS

team goal such as formation. A command view is developed to interact with UAVs. The command view composes of two parts: one is the command issue window, the other is the UAV acknowledgment window. The GCS operator can issue a command and send to UAV directly by typing in the command issue window. Once the command is received by the UAV, an acknowledgment message is sent back from UAV to indicate that this command is successfully received. Given different flight tasks, a command list is developed to fulfill the purpose of preflight testing and practical mission planning on the spot. For example, when the UAV onboard program starts running, a prompt message will be received and displayed "System started, type 'run' to start program, 'quit' to quit system." Once pilot issues the "run" command, the onboard tasking thread will start to execute. All the information view will begin to update every one second. After a while, if all the sensor data display are correct, the ground pilot can issue automatic task commands, such as "hold" to test autonomous hover function. Furthermore, if the autonomous hovering performance is good, waypoints path tracking command can be executed until the task mission is accomplished.

5.6 Conclusion

In this chapter, a MFC based user-friendly ground control system is designed on a Windows based laptop. The communications with multiple UAVs are achieved with sending and receiving capabilities. Various monitoring views are also designed which render the GCS operator a better understanding of the current UAV status during the flight. Task management module also facilitates various user tests with flexible user commands.

Chapter 6

Communications Systems

6.1 Introduction

Another essential part of the UAV framework in Fig. 2.2 is the *wireless communications*. It is via the communications function that the interactions among UAVs and GCS are reachable. As shown in the framework, a UAV needs not only to speak to GCS, it also needs to communicate with other UAVs to realize cooperative tasks. With UAV to GCS communications, the GCS operator can realize remote control and monitor of UAVs in the air. With UAV to UAV communications, the UAV team can multiply their capabilites and effectiveness. To establish a communication network, we should first determine the network architecture given the applications. Then the hardware modules supporting the architecture are selected. Next, we design the communication data format to unify the data exchanged in the network. Finally, the software implementation is conducted.

6.2 Architecture of Communication Network

The communication architecture refers to the configuration of the network communication devices and their operation principles. The configuration of network communication devices (also called network node) can be classified as two categories: centralized and peer-to-peer

(P2P) [69]. In centralized configuration, all network nodes cannot communication with one another directly. For example, if node A wants to speak to node B, node A must first transfer the data to the central node, then central node relays the data to node B. The advantage of this configuration is that central node has all the information available in the network. Another issue is the scalability problem with number of network nodes increase. On the other hand, this configuration introduces unnecessary communication delay. Another approach is the P2P architecture which is distributed, where all nodes can be a client and a server.

Considering the practical communication requirements in multiple-UAV applications, the UAV to GCS communication range should be long enough to cover a few kilometers while the data bandwidth is not intensive. As such, given the specifications in Table 6.1, the UART modem can fulfill the UAV to GCS communication. For UAV to UAV communications, considering the communication range is around 100 m at most during our daily practical flight, the flexible P2P network architecture is thus suitable for onboard inter-UAV information exchange. Equipped with the COTS 802.11 device, each UAV can talk freely to its team members during the flight. Finally, to realize long range video transmission, a 3G modem is deployed both on UAV and GCS to establish telecommunications link to downstream onboard live camera images to GCS. The communication modules deployed onboard are configurable given different applications. For single UAV based flight tasks, only UART modem is deployed. The WiFi module is activated when multiple-UAV cooperative application is required. Also, if onboard live images are needed for monitoring, then 3G modem is mounted on the vision subsystem. Fig. 6.1 demonstrates this hybrid and flexible communication architecture. As illustrated in the figure, the data exchanged among UAVs and GCS can be classified as four categories: 1) flight data, 2) command data, 3) cooperative data, 4) video data. In summary, UAV sends its flight data to GCS for monitoring, GCS operator sends user commands to UAVs. In cooperative scenarios, the cooperative data is exchanged among UAV team members. Lastly, the video data represents the downlink of

image data from UAV to GCS.

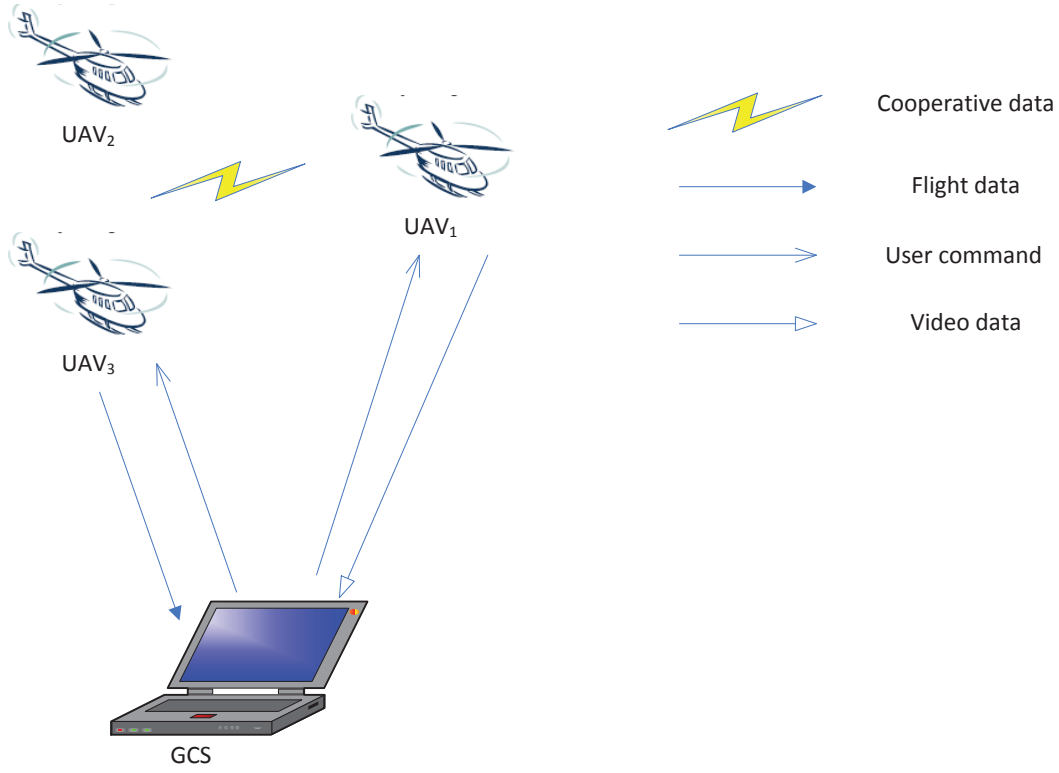


Figure 6.1: Communication architecture in multiple-UAV systems

6.3 Communication Modules

In the multiple-UAV systems, communication range and data bandwidth are two major factors in selecting network modules. As introduced in the last section, three kinds of communication modules are seleted: one is the UART based wireless modem, another one is the 802.11 (so called WiFi [26]) and the last one is the 3G modem. We select the FreeWave IM-500 (see Fig. 6.2) as the UART based modem. It works at 2.4 GHz and long working range of up to 32 km in open field. It supports a baudrate up to 115.2 Kbps which provides sufficient bandwidth for downloading UAV status data. It supports centralized working

Figure 6.2: UART based wireless modem - FreeWave IM-500

architecture, where only one master modem and multiple slave modems can be used to construct the wireless network. For the hardware interface of WiFi, we select an industry stand module, the ACKSYS WLg-LINK-OEM shown in Fig. 6.3 to fulfill the peer to peer wireless communications. This module features an Ethernet interface and requires no special drivers in the QNX RTOS.

The 3G modem adopts a mobile telecommunications technology with the radio stations infrastructure to realize seamless, high bandwidth communication anywhere in the world. The typical transmission rate is around 300 kbps during mobile movement such as UAV in the flight. For really long range UAV flight such as tens of kilometers, the 3G communication mechanism is so far the best solution to offer both long range and high transmission rate in the commercial market. The Telit UC864-E [65] as shown in Fig. 6.4 is selected in our system to realize 3G communications. The main specifications of the three communication modules are summarized in Table 6.1.

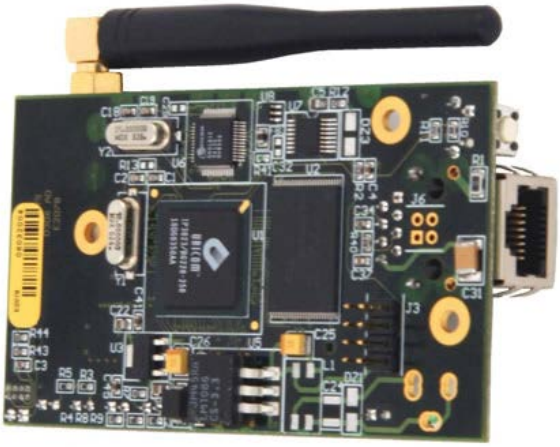


Figure 6.3: WiFi module - ACKSYS WLg-LINK-OEM

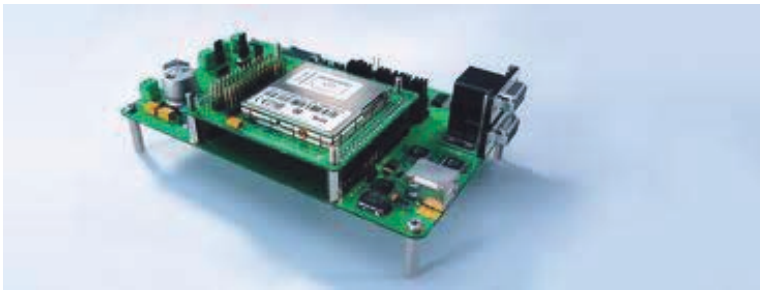


Figure 6.4: 3G communication module - UC864

Table 6.1: Communication device specifications

Communication module	Data bandwidth	Range	Protocol
IM-500	115200 bps	32 km	UART
AckSys	11 Mbps	200 m	TCP/IP
Telit UC864-E	200 kbps	Hundred km	TCP/IP

Table 6.2: Communication data format

Data field	Definition
from (unsigned char, 1)	Sender ID
to (unsigned char, 1)	Receiver ID
size (short, 2)	Total bytes of packet
code (short, 2)	The code representing packet type
time (double, 8)	The time when packet sent
data	The data payload given a packet
checksum (short, 2)	The checksum of data fields including code, time and data payload

6.4 Communication Protocol

Next, the operation data format needs to be designed such that both sender and receiver can understand each other. Given the fact that different types of data needs to be transmitted, a universal type of data packet is designed to accommodate various types of packages. The communication protocol is designed as shown in Table 6.2.

With the above data format, another data structure *TELEGRAPH* is adopted to encapsulate the data as a telegraph to be sent.

```
struct TELEGRAPH {
    int size;
    char content[512];
};
```

For example, given a flight status data to be sent from UAV to GCS, the data payload should be a data variable with the structure of *UAVSTATE* as introduced in Section 3.3. The size of the telegraph to be sent is 232 bytes in total, where 216 is the size of *UAVSTATE* (total 27 variables with 8 bytes double format representation). Then this telegraph can be sent via the sending operation of UART.

$$1 + 1 + 2 + 2 + 8 + 216 + 2 = 232$$

6.5 Software Implementation

The software implementation of communications involves the device configuration, sending and receiving functions. With these three operations, the *Communications* software module can be designed to be portable for other communication hardware modules. The configuration is device specific, while read and write operations of UART and socket are standard function calls provided in the operating system. As explained in Section 3.9, the sending and receiving operations are designed as two separate working threads. The sending thread *clsCMM(TX)* is activated periodically while receiving thread *clsCMM(RX)* is activated only when data is received on the hardware module. As we have three kinds of communication modules, a set of sending and receiving task threads are designed for each hardware module respectively.

6.5.1 Network Configurations

The configuration mainly refers to the network configuration of UART modem, WiFi module and 3G modem to realize the proposed hybrid architecture to realize the corresponding communication links described in Fig. 6.1. Given the manual [44], the UART modem can be configured as one master node at GCS, and other slave nodes at each UAV. In this case, the GCS can send commands to a certain UAV or the whole UAV team. Meanwhile, each UAV can only send its flight data to GCS. To realize the inter-UAV communication, the WiFi modules are configured as Ad-Hoc mode and manually assigned the IP address for each member. With such two network architectures combined, both long range master-slave and P2P cooperative communications can be achieved. On vision subsystem, the 3G modem needs to get access from the telecommunication service provider and thus be able to connect with Internet. Similarly, the GCS side also needs a 3G modem to establish a point-to-point connection with the UAV vision subsystem.

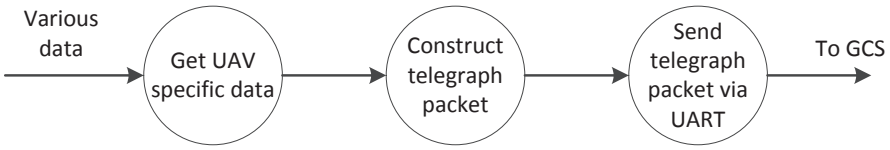


Figure 6.5: Data flow diagram of sending

6.5.2 Sending Operation

The sending operation on onboard system is the function of the class *clsCMM* as explained in Section 3.9. As UAV needs to send its flight data including status data *UAVSTATE*, D/A data *DAQDATA*, manual control signal *SVODATA*, automatic control signal *SIGDATA*, periodically to GCS, the execution frequency of sending operation can be set as 1 Hz as explained in Section 3.9. The onboard main loop execution frequency is set as 50 Hz, then *clsCMM* will perform sending every 50 loops. The data flow diagram modeling the sending operation is shown in Fig. 6.5. First, the sending thread will get the various types of data by retrieving corresponding data from other class objects by method *Get UAV specific data*, then encapsulate this data structure into the telegraph structure *TELEGRAPH* with *Construct telegraph packet* and finally send out this packet with the sending operation *Send telegraph packet via UART*. The encapsulation process is performed based on the communication protocol as listed in Table 6.2. When the sending thread is activated, the described data flow is executed for each kind of data including *UAVSTATUS*, *DAQDATA*, *SVODATA* and *SIGDATA*.

Regarding the socket based sending operation, another task thread *clsNET(TX)* is implemented with similar data flow mechanism as shown in Fig. 6.5. But the data encapsulated is the cooperative packet to realize multiple-UAV formation task. The data format is introduced in Section 7.3.

For GCS side, the sending operation also applies. In this case, user command is encapsulated in the packet instead of various types of specific data. The code in Table 6.2

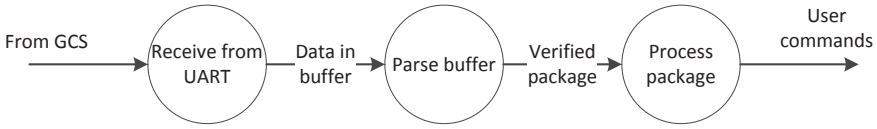


Figure 6.6: Data flow diagram of receiving

is replaced with command ID. As various commands as listed in Table 3.1 are needed for GCS operators, a data structure *COMMAND* is used to represent the command type and its corresponding parameters if any.

6.5.3 Receiving Operation

The receiving operation is assigned as a background task which is activated once new data arrives. The data flow diagram of receiving procedure is described in Fig. 6.6. The onboard system first copy the received data to memory for processing, then verify the packet given the data format with *Parse buffer*, then the verified packet is parsed to extract the command code and its parameters.

As shown in the figure, the function *Parse buffer* is for parsing the received byte one by one until a verified package is extracted. This tedious procedure can be described with the state transition diagram as shown in Fig. 6.7. The *parse buffer* operation is analyzed based on the data format. The data field of *from*, *to*, *size* and *checksum* are examined sequentially in each state. If any incorrect bytes are detected, then the state will transit to the first one. If every data field is correct, then a verified package can be extracted out finally. This procedure iterates until all the data bytes are analyzed.

With the verified package, further analysis is necessary to extract user command code and parameters. An acknowledgment message is replied to GCS operator to indicate the successful receiving. This procedure can be described with the data flow diagram presented in Fig. 6.8.

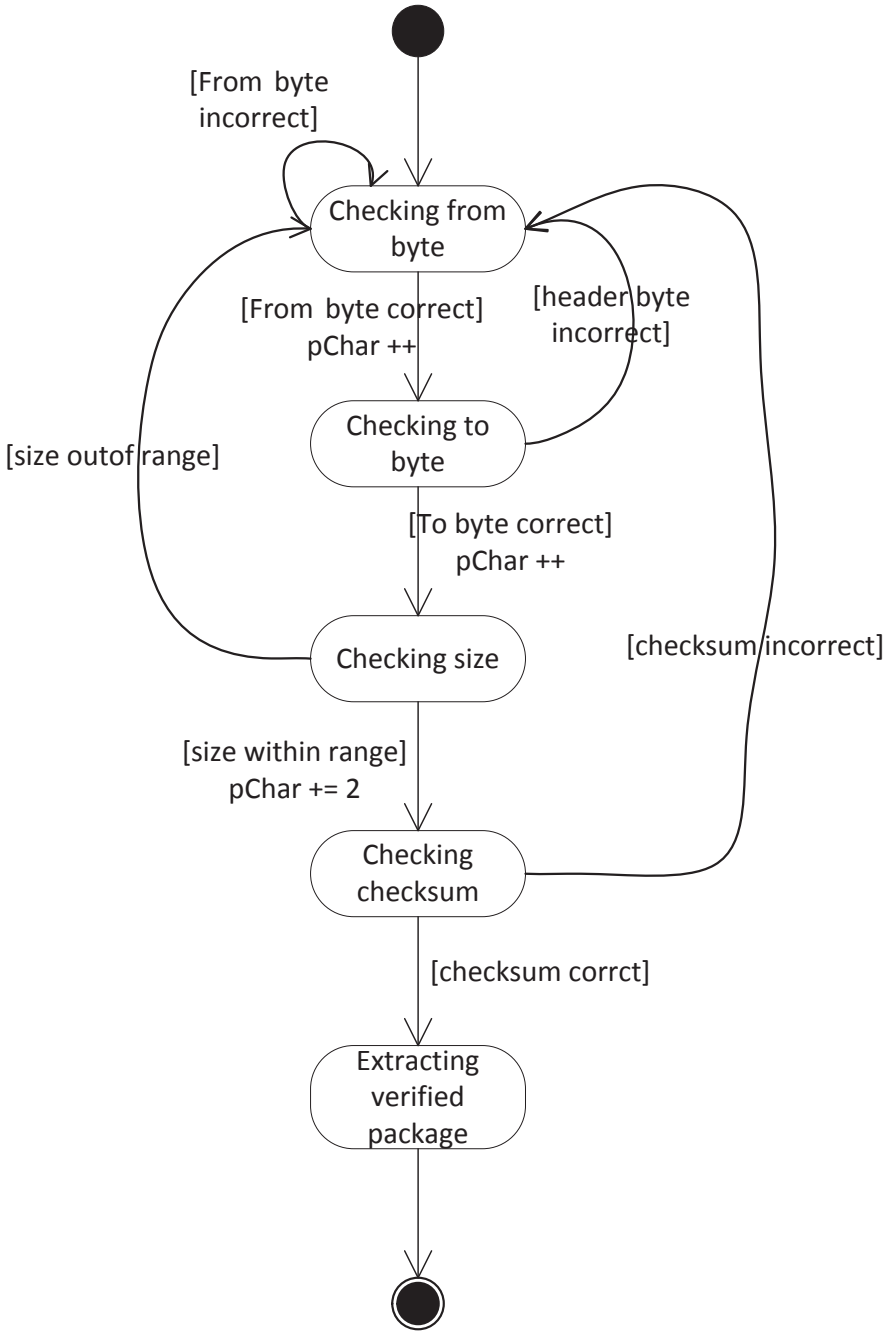


Figure 6.7: State transition diagram of parsing buffer

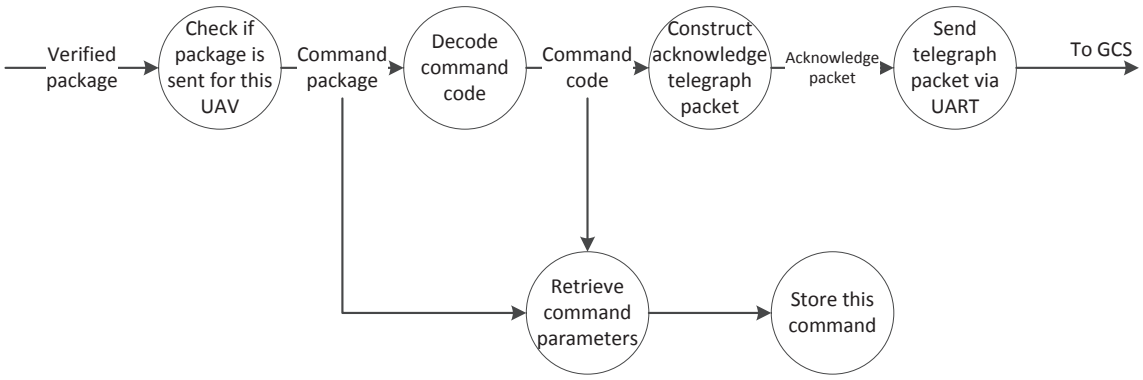


Figure 6.8: Data flow diagram of processing package

6.5.4 Software Modeling

With the above specification decomposition of sending and receiving with data flow diagrams, the class diagram of communications *clsCMM* is drawn in Fig. 6.9 to piece together the properties and bebaviors that should be implemented. Please note that both sending and receiving functions are incorporated into this class, though the execution of each function is scheduled by the main program introduced in Section 3.9. Because the data flow diagrams can apply to WiFi module as well, the functions of UART can be deployed for WiFi communications as well.

6.6 3G Communication

The video stream transmission adopts the novel 3G network which possesses really long range distance and high data bandwidth (around 200 kpbs). To facilitate the image transmission, the images are further compressed via JPEG compression method provided by the OpenCV library. The final compressed image size is around 3K Bytes, which can be reasonably accommodated by the 3G network bandwidth. The update frequency is adopted as 2 Hz in the current system.

The communication protocols for image transmission is initially selected as TCP which

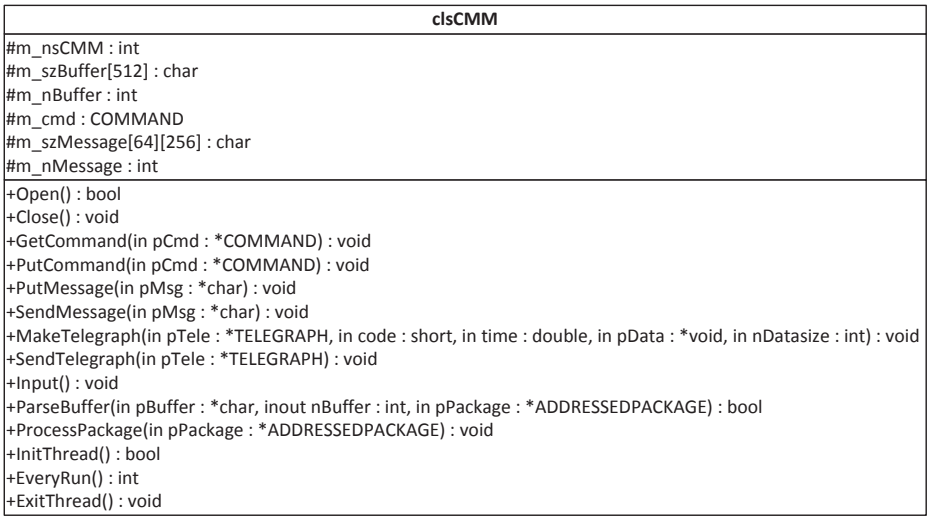


Figure 6.9: Class diagram of clsCMM

can guarantee reliable image transmission though some delays can be expected for the handshaking, image retransmission. The image transmission data format are designed as shown in Fig. 6.10. The image transmission from onboard to GCS involves two steps: first, the information of the image *IMAGEHEADER* is sent to GCS, the GCS will perform the preparation work given the image type and image size. Second, the data bytes in the image is encapsulated in the designed format and sent to GCS and thus finish the transmission of one image. As TCP is selected as the transmission layer protocol, the image will be displayed on the GCS reliably but with a certain amount of delay. The delay is due to the large traffic volume in the cellular 3G network, where a lot of packets other than the onboard image data are stored and relayed. While in a WiFi environment, the image transmission rate can reach up to 10 Hz or higher.

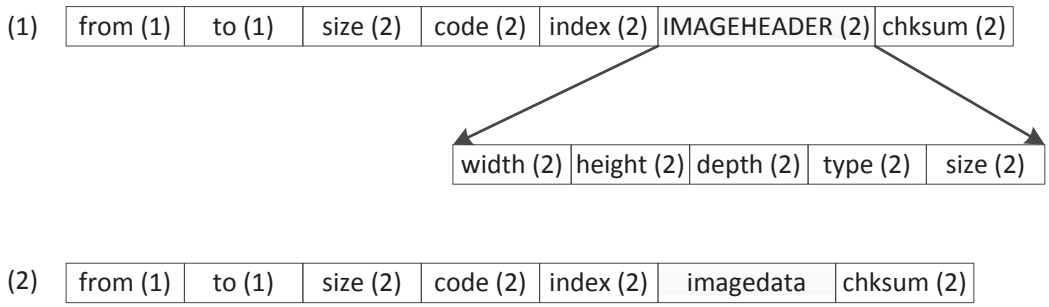


Figure 6.10: Image communication data format protocol.

6.7 Conclusion

In this chapter, the hybrid communications network is established by adopting UART based modems and WiFi modules. With this hybrid configuration, the UAV team can realize long range communications with GCS and inter-UAV cooperative information exchange. The sending and receiving function on both UAV and GCS are modeled with the data flow diagrams. Correspondingly, the class diagrams of communcation task threads are presented. The video transmission based on 3G is also implemented.

Chapter 7

Experimental Results and Applications

7.1 Introduction

The proposed software system has been successfully applied on different UAV applications to verify its universal and configurable features. First, the software system has been deployed on the applications on a single UAV, such as HeLion and SheLion, to realize fully envelop flight control, GPS-based waypoint navigation, vision-based target tracking, etc. To demonstrate the expansion capability of the software system, it has been utilized to the applications of multiple UAV collaboration, including leader-follower based formation, hybrid formation reconfiguration, etc. Moreover, to further verify the robustness and efficiency of the software system, it has been exported to an unconventional coaxial UAV, GremLion, for more practical and demanding tasks required by the UAVForge [66] competition held by DARPA in USA on May, 2012. In the following parts, various practical applications are presented.

7.2 Automatic Flight

The fundamental task of a UAV is to be capable of fully autonomous flight. To effectively test the overall performance of onboard system and GCS, two kinds of tests has been conducted. One is the experiment of the behavior-based control mechanism by using a full envelop flight. Another one is the test of reliability and stability by using aggressive behaviors, a fast forward flight with a speed up to 12 m/s.

7.2.1 Full Envelop Flight

The full envelop flight is composed of various scheduled behaviors such as take-off, hover, trajectory tracking, landing and etc. The scheduled flight designed is shown in Fig. 7.1. It consists of the following seven scheduled tasks, including:

1. Takeoff: The UAV is commanded to start its engine from low to hight and lift from ground from starting position (marked as 'O') to point A, which is 15 m above the ground.

2. Slithering: From point A, the UAV moves in a zigzag path to point B. The distance covered between A and B is 60 m.

3. Head turning: The UAV is scheduled to flight from B to A. Meanwhile, its heading is rotating during movement.

4. Pirouetting: The UAV tracks a circle with a radius of 10 m with heading pointing to the center of the circle.

5. Vertical wheeling: The UAV is commanded to fly a circle with a radius of 10 m in the vertical plane. Star from point A, it will fly a one and half circle.

6. Downward spiraling: The UAV flies along a downward spiral path in the backward direction. Then the UAV returns to point A.

7. Landing: The UAV descends to the ground from point A, and shut down the engine.

The behaviors of the scheduled flight and response of position are provided in Fig. 7.2. As labeled in the figure, the behaviors including enginelow, engineup, takeoff, path, landing

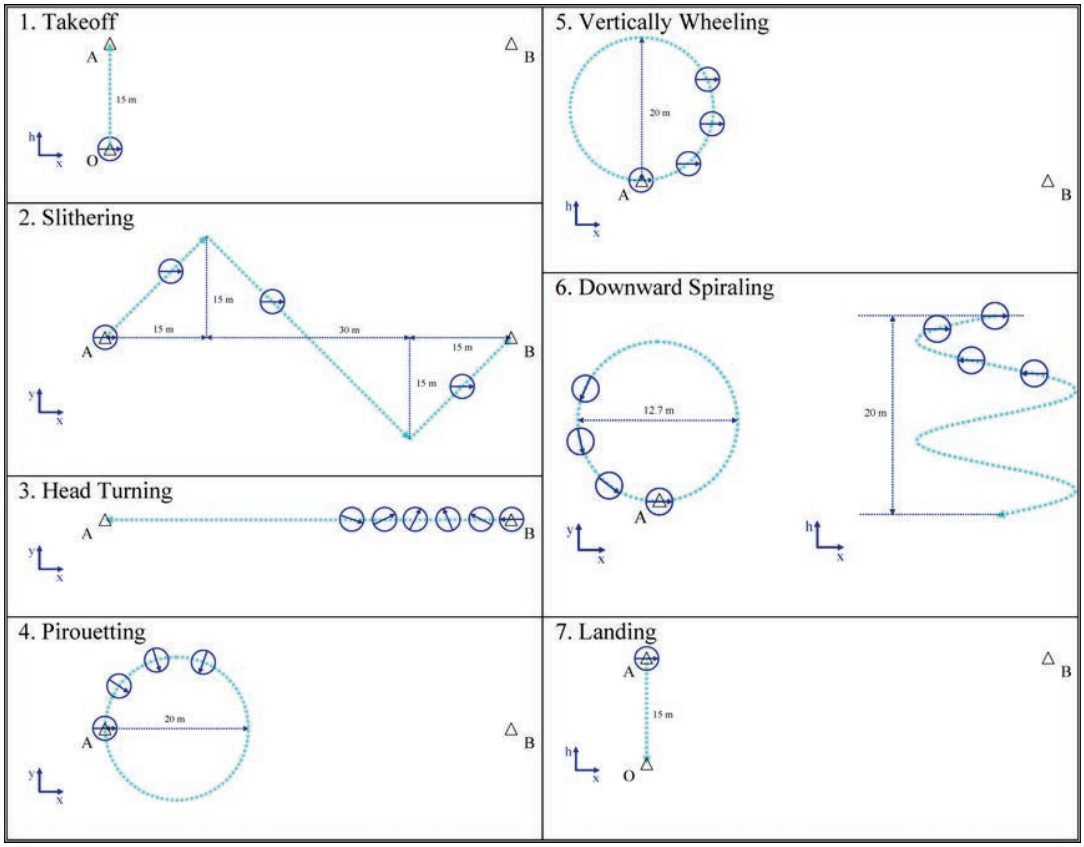


Figure 7.1: Task schedule of full envelop flight

and enginedown are triggered correctly as scheduled. The position tracking response of each path in Fig. 7.3 during the full envelop also verify the successful design of the behavior-based control methodology.

7.2.2 Fast Flight

In this experiment, an aggressive flight with a fast forward speed of 12 m/s is conducted [46]. The UAV accelerates gradually to 12 m/s and flies forward with a distance of around 200 m. This aggressive flight also incorporates several automatic behaviors such as *Forward flight*, *Hover*, *Backward flight*, *Hovering turn*, *Vertical maneuver*, *Lateral reposition*, *Turn-to-target*, *Slalom* and *Pirouette* [4]. In implementation, we concatenate all the behaviors

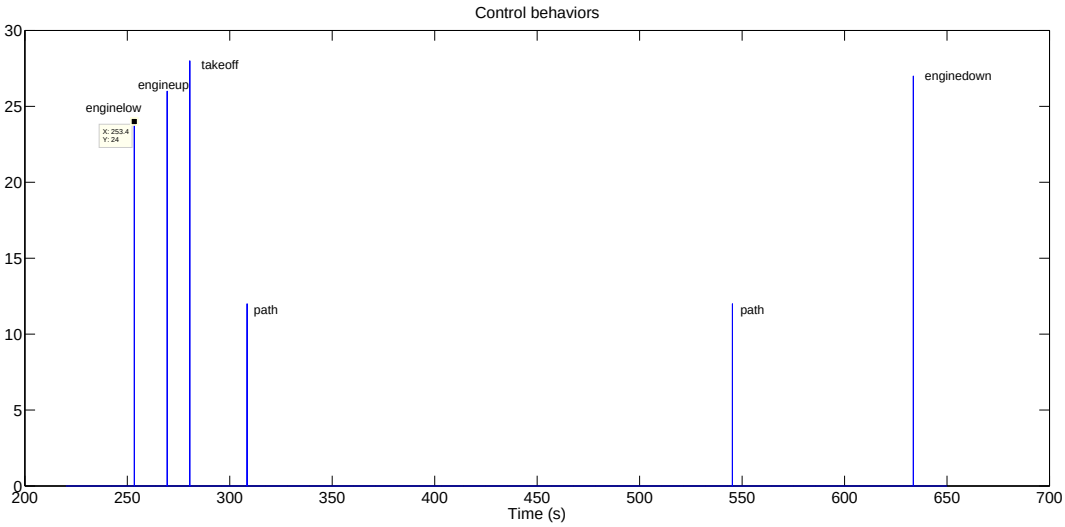


Figure 7.2: Behaviors in full envelop flight

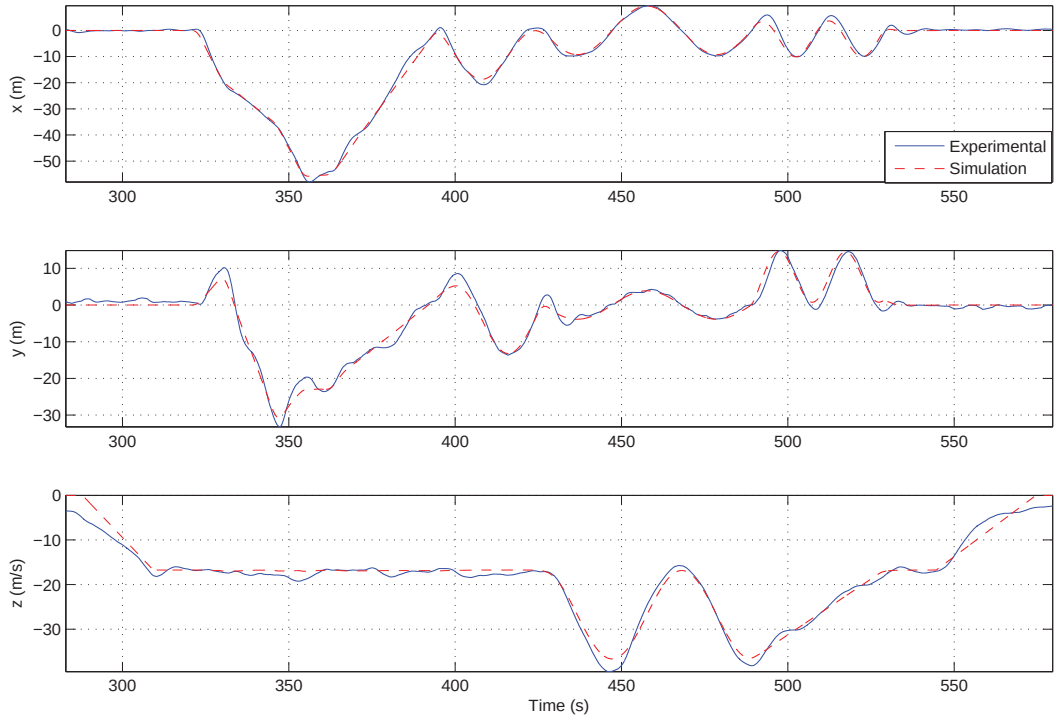


Figure 7.3: Response of full envelop flight

with the pre-defined paths. The flight results are shown in Fig. 7.4. Due to the fast velocity, several linear models given different flight conditions are derived based on the nonlinear model. Correspondingly, the H_∞ control laws and trim values are calculated given the conditions. More details can be found in [4]. As shown in the figure, the forward velocity perfectly tracks the reference which proves the control performance implemented in the onboard software is reliable even under the situations with fast movements.

7.3 Formation Flight

7.3.1 Software Design

With good performance of the single UAV automatic flight task, the multiple-UAV application can be carried out. In formation flight, a leader-follower based approach is applied in our system [45, 16, 28]. The follower will keep a certain distance away from the leader when the leader is performing a path tracking.

For the case of formation flight, the leader is commanded to perform a predefined path tracking, and the follower is required to follow the leader with a constant distance offset along both longitudinal and lateral directions in the coordinate of the leader. HeLion is assigned as the leader, and SheLion performs as the follower. Various scenarios have been evaluated and here we adopt a circle path tracking as the example, which is shown in Fig. 7.5. Note that L_0 and F_0 are the initial reference rendezvous positions for the leader and the follower, respectively. The points L_i $(i = 1, 2, ..., N)$ refer to the predefined trajectory for the leader, and the points F_i $(i = 1, 2, ..., N)$ refer to the reference points that the follower should track provided by the formation algorithms.

It is clear that the information exchange is mandatory in the multiple-UAV formation application. The P2P network architecture is deployed in formation flight considering reconfigurability and scalability. A communication mechanism has been proposed to realize the P2P network architecture, which is illustrated in the message sequence diagram in Fig. 7.6.

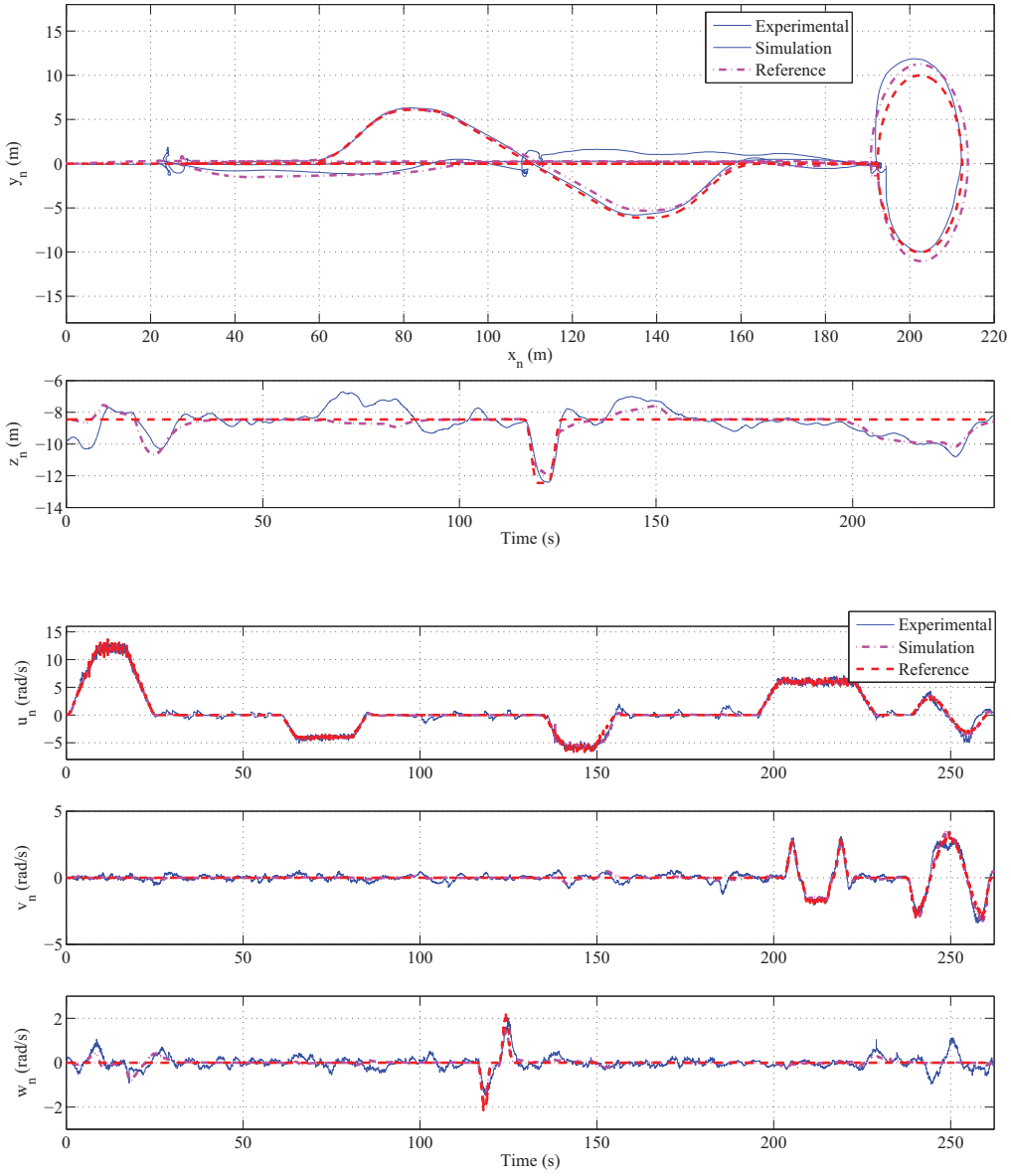


Figure 7.4: Responses of fast forward flight - position and velocity

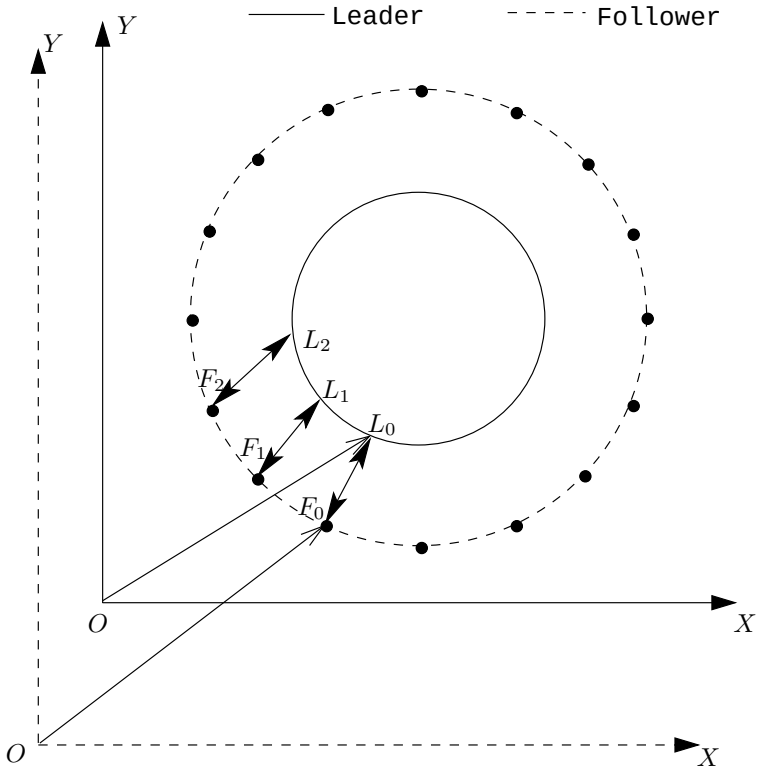


Figure 7.5: Leader-follower circle formation scenario

In order to facilitate cooperative data exchange, a communication protocol is designed to fulfill the cooperative task as shown in Table 7.1. This cooperative data packet information can be adaptable given different cooperative tasks. In formation flight, the cooperative packet contains the handshake messages between the leader and the follower for reliability of inter-UAV communication. Once handshake is successful, the leader will update its states to the follower continuously during the flight. When the leader receives the "formation" command from GCS, it sends the handshake packet *CONNECT_INIT_LD* with its initial GPS information to the follower with the packet data as shown in Table 7.2. If the follower correctly receives the packet from leader and identifies it is a *CONNECT_INIT_LD* packet, it will reply with *CONNECT_ACK_FL* packet as shown in Table 7.3. Once leader receives the acknowledgment packet from the follower, it starts sending update packet *LEADERFOR-*

Table 7.1: Cooperative data packet format.

Data packet field	Definition
type (int)	type of this packet
role (int)	role of the sender
no. (int)	packet sequence number
x (double)	longitudinal position in NED frame
y (double)	lateral position in NED frame
z (double)	vertical position in NED frame
c (double)	heading angle
u (double)	longitudinal velocity in NED frame
v (double)	lateral velocity in NED frame
w (double)	vertical velocity in NED frame
lat (double)	initial latitude
lon (double)	initial longitude

MATION_UPDATE with position, heading and velocity to the follower with a frequency of 10 Hz shown in Table 7.4. Meanwhile, the follower replies each packet with the type of *COOP_REPLY_FL* and the packet sequence number of the leader as shown in Table 7.5.

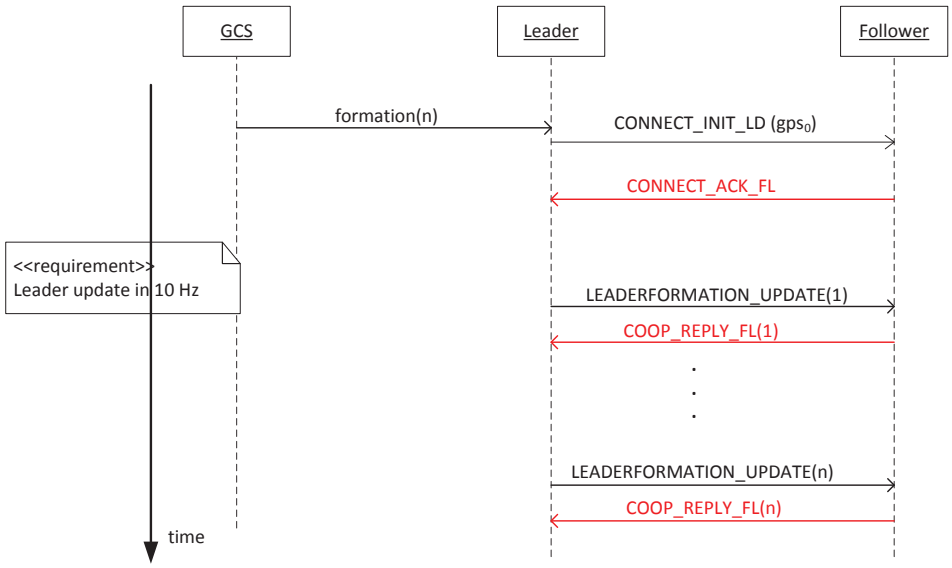


Figure 7.6: Message sequence diagram in formation flight.

Table 7.2: Cooperative data packet format of leader to initiate connection

Data packet field	Value
type (int)	CONNECT_INIT_LD
role (int)	LEADER
no. (int)	0
x (double)	0
y (double)	0
z (double)	0
c (double)	0
u (double)	0
v (double)	0
w (double)	0
lat (double)	initial latitude
lon (double)	initial longitude

Table 7.3: Cooperative data packet format of follower to acknowledge connection

Data packet field	Value
type (int)	CONNECT_ACK_FL
role (int)	FOLLOWER
no. (int)	0
x (double)	0
y (double)	0
z (double)	0
c (double)	0
u (double)	0
v (double)	0
w (double)	0
lat (double)	initial latitude
lon (double)	initial longitude

Table 7.4: Cooperative data packet format of leader update

Data packet field	Value
type (int)	LEADERINFORMATION_UPDATE
role (int)	LEADER
no. (int)	n
x (double)	current position in NED x-axis x
y (double)	current position in NED y-axis y
z (double)	current height in NED z
c (double)	current heading in NED c
u (double)	current velocity in NED x-axis u
v (double)	current velocity in NED y-axis v
w (double)	current velocity in NED z-axis w
lat (double)	initial latitude
lon (double)	initial longitude

Table 7.5: Cooperative data packet format of follower to reply leader update

Data packet field	Value
type (int)	COOP_REPLY_FL
role (int)	FOLLOWER
no. (int)	n
x (double)	0
y (double)	0
z (double)	0
c (double)	0
u (double)	0
v (double)	0
w (double)	0
lat (double)	0
lon (double)	0

Next, a cooperative software module is designed to achieve the aforementioned requirements. This formation module can be deployed on any kind of UAV and can be assigned as either a leader or a follower without bothering the bottom details. This formation module is described in the state transition diagram as shown in Fig. 7.7. Once the UAV receives the cooperative packet, a process method for leader and follower are executed based on the role of this packet. Both leader and follower will process the packet type based on the communication protocol in Fig. 7.6. As this module is deployed exactly the same in each UAV, the flexibility of assigning the role of UAV members can be easily achieved. Once formation task is issued from GCS, a new behavior, the formation behavior is activated on the leader and the follower. Based on the control structure in Fig. 3.9, the cooperative packet receiving function can be integrated into the *From wireless communications*. The formation algorithm can be put into the high level *Scheduling* block. Correspondingly, the data flow diagram of the flight control is redrawn in Fig. 7.8 to add the formation processing module. We notice that another *Formation reference generation* is inserted in parallel with the other reference generation. Similarly, the reference source is determined by the *Reference selection* based on the behaviors extracted from user commands. A "formation(n)" command from GCS operator will enable this reference in the control loop.

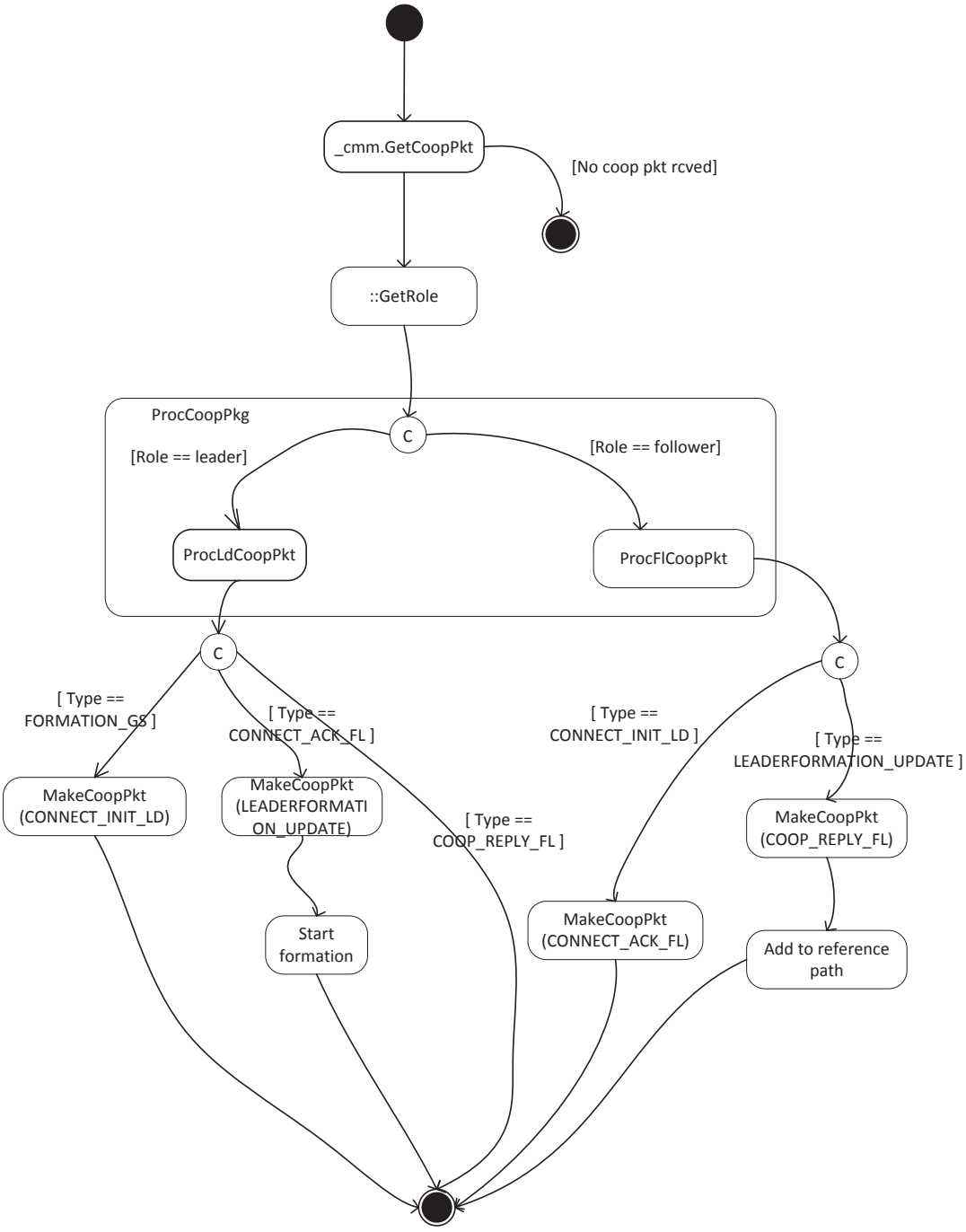


Figure 7.7: State transition diagram in formation flight

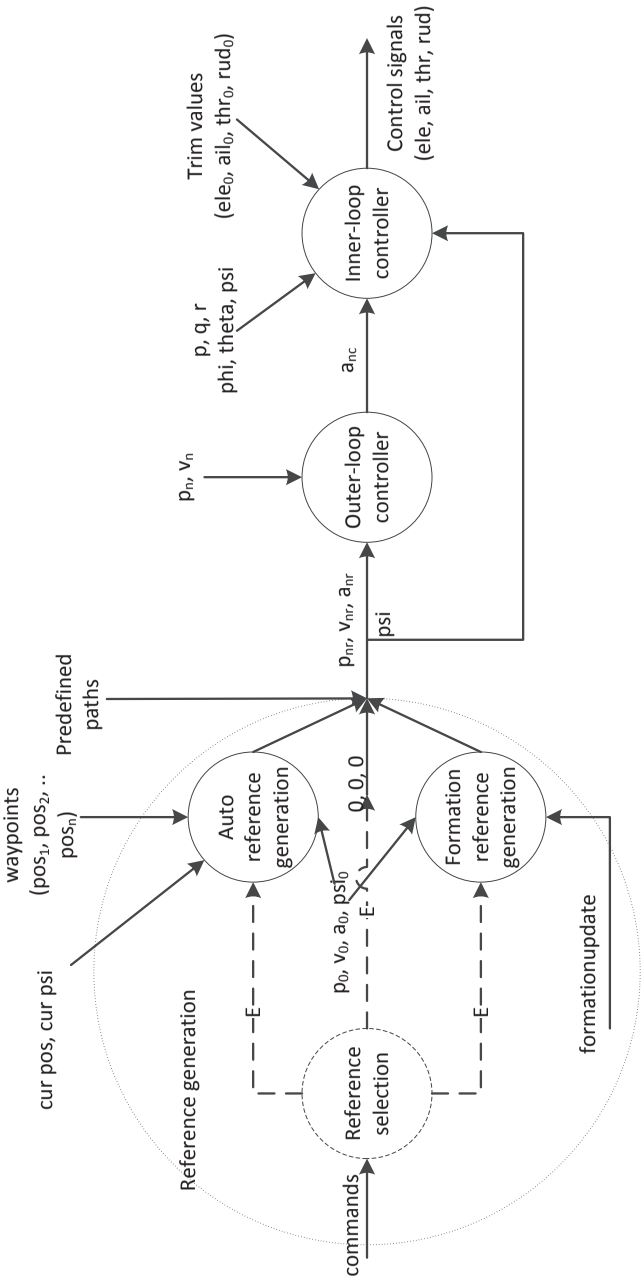


Figure 7.8: Data flow diagram of control in formation flight

7.3.2 Formation Flight Results

The simulation results are depicted in Fig. 7.9 to 7.10. In the simulation, the initial starting point of the leader and follower can be determined on the Google Maps view. The information exchange frequency is set as 10 Hz and the leader is commanded to perform a circle path tracking with a radius of 10 m with an tangential velocity of 1 m/s. It is observed that in Fig. 7.10, the follower tracks closely the reference generated based on leader update. We also notice that since the distance between the leader and the follower is less than the required distance at first, the follower performs a rendezvous task such that two UAVs are ready for the formation task. Fig. 7.10 also indicates that the heading angle tracking is also sufficiently accurate. Such simulation results provide us with enough confidence to conduct the formation flight practically.

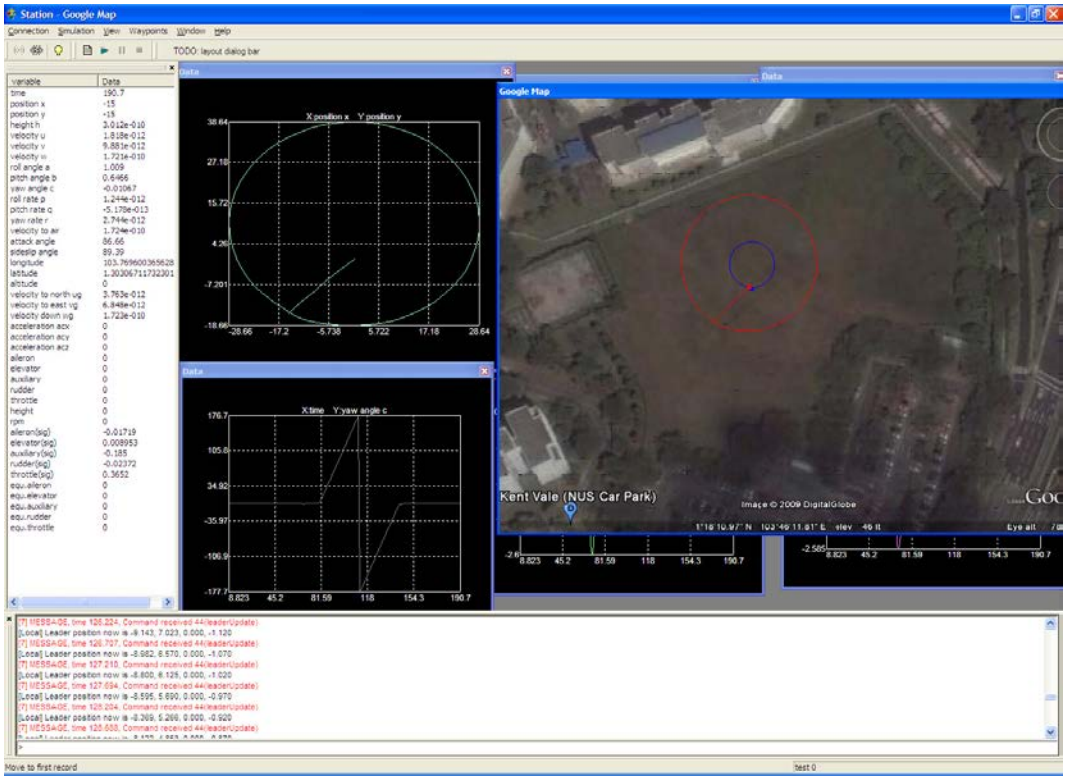


Figure 7.9: Screenshot of leader-follower formation in the GCS

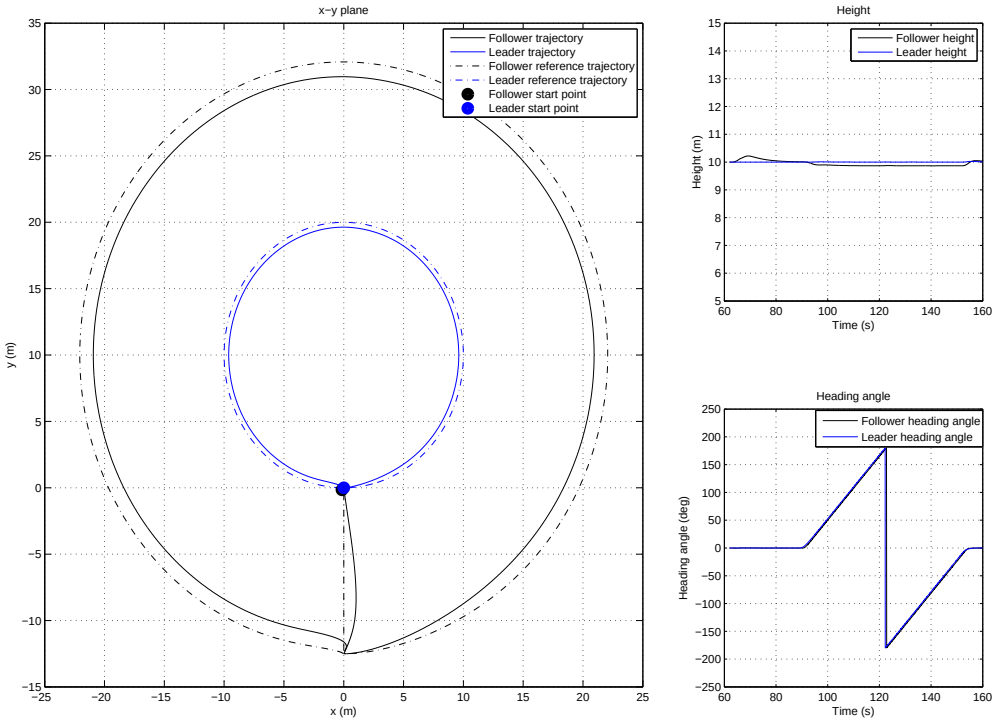


Figure 7.10: Leader-follower in a circle formation

After finishing the successful simulation, we proceed to the practical implementation stage. Through the comprehensive literature survey, we realize that the flight formation of UAVs leaded by UAV is an equivalently important issue and having a trend to take the leading role in the practical implementation field of formation flight. As such, we have further extended our formation control methodology to this area and the implementation results are greatly successful and impressive. In what follows of this section, we will detail the experimental results and associated analysis for both the MAV-lead and UAV-lead formation flight.

MAV-lead Formation Flight

The formation flight of a UAV helicopter leaded by an MAV helicopter has various advantages. The most distinguished character is that it can make full use of the intelligence and flexibility of piloted control and the unique properties of UAVs (such as ultimate precision and hazard immunity), and greatly improve the possibility and feasibility to complete designated missions.

In this experiment, the ground pilot controls the leader UAV to fly with a text shape of "NUS". The inter-UAV communication follows the information exchange protocol mentioned before. The two UAVs are controlled in the hover status first. Once GCS receives the formation acknowledge message of the follower, the pilot can start slowly moving the leader to conduct the "NUS" track. It is expected that the follower can follow the leader with a lateral distance of 10 m perfectly. The final MAV-lead formation flight result is shown in Fig. 7.11, where the blue curve is the 3D trajectory of the leader, and the red curve of the follower tracks the leader very well.

UAV-lead Formation Flight

Although the formation flight of the UAV helicopter leaded by an MAV helicopter has been completed with a complex trajectory, we have realized that it is not as complete as we expected, from the point of view of practical implementation. Due to various reasons such as (1) the limited range of radio control, (2) the hazardous environment where the human pilots can not enter, and (3) the long time but continuous flight which is difficult for human pilot to endure, the formation flight of UAVs leaded by MAV may not be implemented in many practical situations. Instead, they can be efficiently completed by the formation flight of a group of UAVs, in which one UAV is selected as the leader. The main drawback of the formation flight of UAVs leaded by a UAV is that it is not as intelligent as the one leaded by an MAV. However, the UAV platforms are immune to many of the harmful environment factors. Another big advantage of the formation flight leaded by a UAV is that the precision

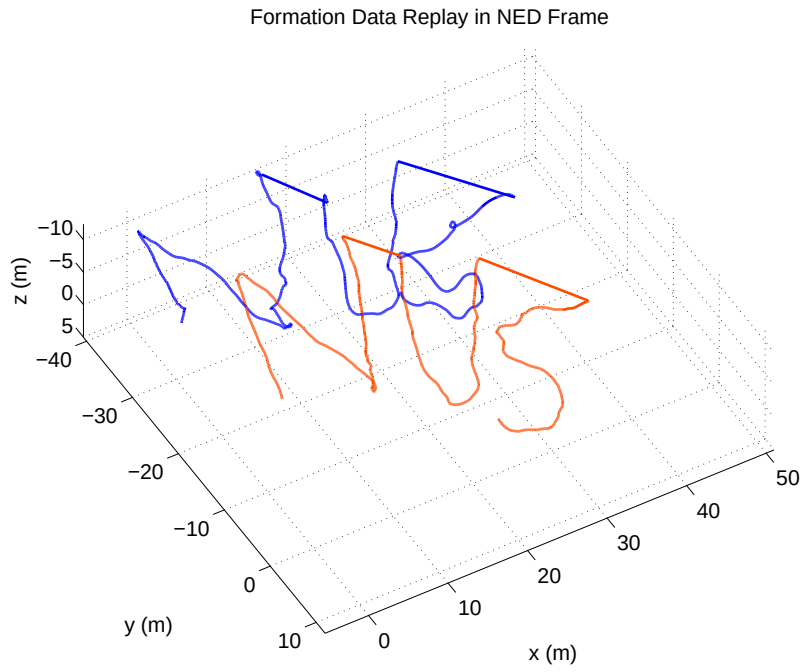


Figure 7.11: MAV-lead formation flight 3D trajectory

and accuracy during the flight can be greatly enhanced. As such, implementing the flight control law in such formation scenario is equivalently important.

To sufficiently evaluate the feasibility of the control law designed for formation flight, we have selected more complicated flight missions for the UAV-lead formation flight. The following representative examples are selected for the illustration purpose, that is, the UAV-lead formation flight following (1) the raceway path, and (2) the head-turning trajectory. In the practical flight tests, the two paths are combined in one single flight to save the time.

The raceway path is a anti-clockwise racetrack trajectory. Substantially, it can be regarded as two straight-line flights, combined with two half concentric-circles flight sections. The leader will keep flying with a tangential speed of 2m/s. The next leader path is the head-turning, which the leader keeps it head turning with a constant rotation speed of 18 $^{\circ}$/s. The perfect 3D trajectory of formation flight is demonstrated in Fig. 7.12. Interested

readers are refereed to our video gallery (http://uav.ece.nus.edu.sg/video.html) for the video recording of the practical flight procedure. Fig. 7.13 is a picture of leader and follower during the formation flight.

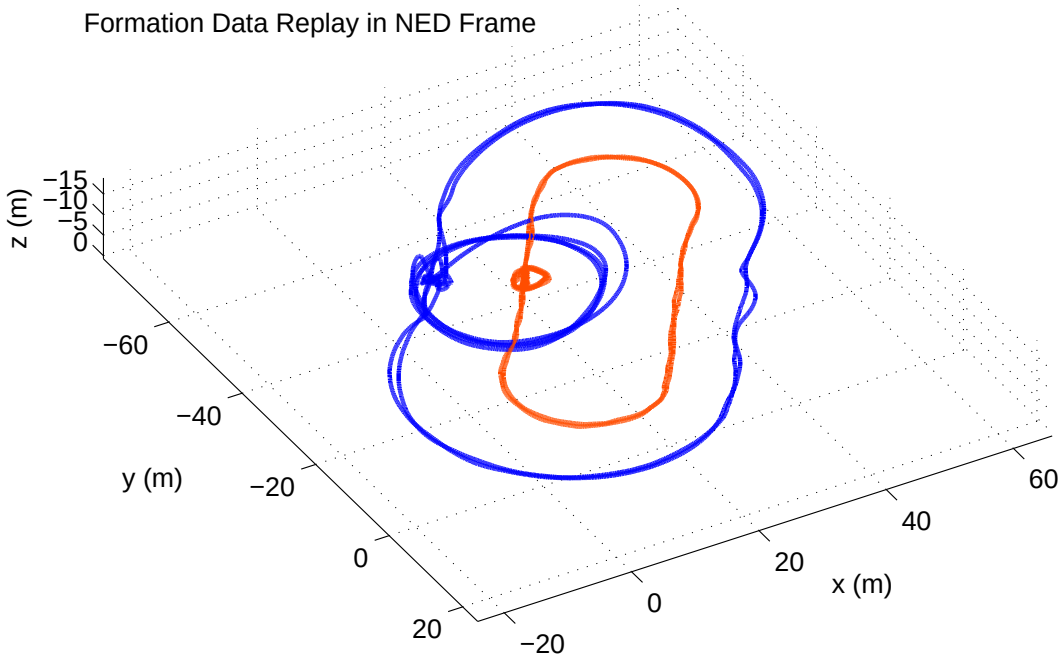


Figure 7.12: UAV-lead formation flight 3D trajectory

7.4 Vision-based Target Detection and Following

The onboard software system consists of the flight control subsystem and vision subsystem. The collaboration of these two subsystems has been realized to perform vision related tasks, such as target detection and tracking. A target detection algorithm has been implemented on the vision subsystem, which is able to autonomously identify a pre-defined ground target in the image. The onboard software provides the capability to synchronize the obtained vision information with state of the UAV to estimate relative distance of the target with respect to the UAV. The vision related work can be referred to [32].

To realize the target tracking, the control behavior of target tracking has been inte-

Figure 7.13: Leader and follow in formation flight

grated into the current control data flow, which is shown in Fig. 3.12. By modifying the reference generation module, the references for target tracking can be deployed from the vision subsystem. The control flow is enriched with a *Target reference generation* added as shown in Fig. 7.14. The target estimation received from the vision subsystem is fed into the outer-loop control law with necessary coordinate transformation.

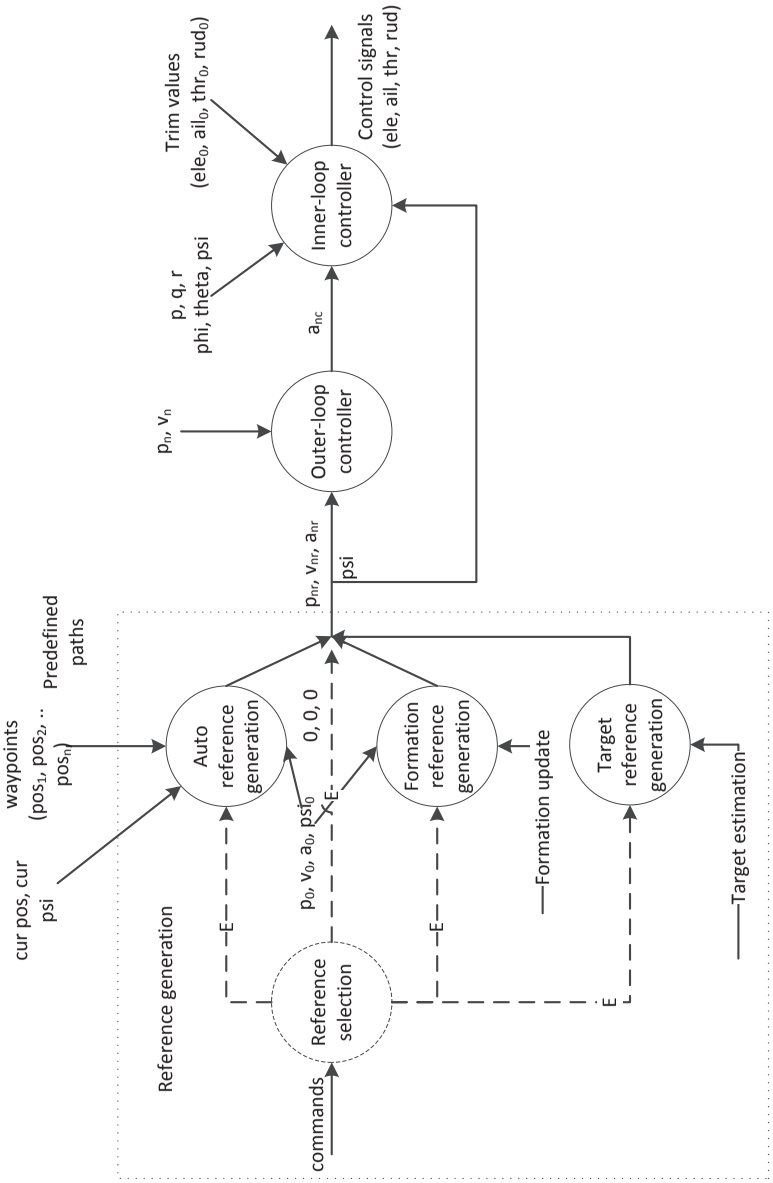


Figure 7.14: Data flow diagram of control in target tracking

During the experiment, SheLion has been adopted as the platform and a toy car is deployed as the ground target. The toy car is moving while SheLion is hovering above it. Once the target is detected and locked by the vision subsystem, GCS operator will issue the "track" command to SheLion and onboard control reference generator will fetch the target estimation data. The target following result is shown in Fig. 7.15, where the UAV follows the onboard generated reference quite well. The onboard reference record also proves the target estimation reference is correctly received and activated as designed in the software.

7.5 UAVForge Competition

Our NUS UAV research team participated the UAVForge competition organized by US DARPA in May 2012 at Georgia. This competition proposed a lot of challenging tasks such as long distance flight to 2 miles, long range video transmission and vision based landing. To fulfill the requirements, we have proposed various approaches to finish the mission. One is to perform all the tasks with pure automatic capabilities. The other is to involve the ground pilot assistance in control based on the video received on GCS. The latter way is more safe and we name it semi-auto mode, in which the GCS operator just controls the velocity while the attitude is still stabilized by onboard inner-loop controller. As such, the GremLion needs to have two modes of flight and can be smoothly switched in the air by GCS operator. As GremLion is a little bit sensitive in the mechanical structure, the semi-auto mode will assist pilot to successfully lift the UAV from the ground.

The above control approach with mode switching is well implemented with the flexible software control structure as shown in Fig. 3.14. Given the data flow diagram, a user command can be incorporated into the current command set as the input to the control bubble *Mode selection* to decide where the reference signal generates from. In semi-auto mode, the joystick input from aileron and elevator are translated to 2D velocity reference in heading frame, while the rudder input is translated to the heading angle reference. The semi-auto flight performance is demonstrated in Fig. 7.16. The result shows that x-axis

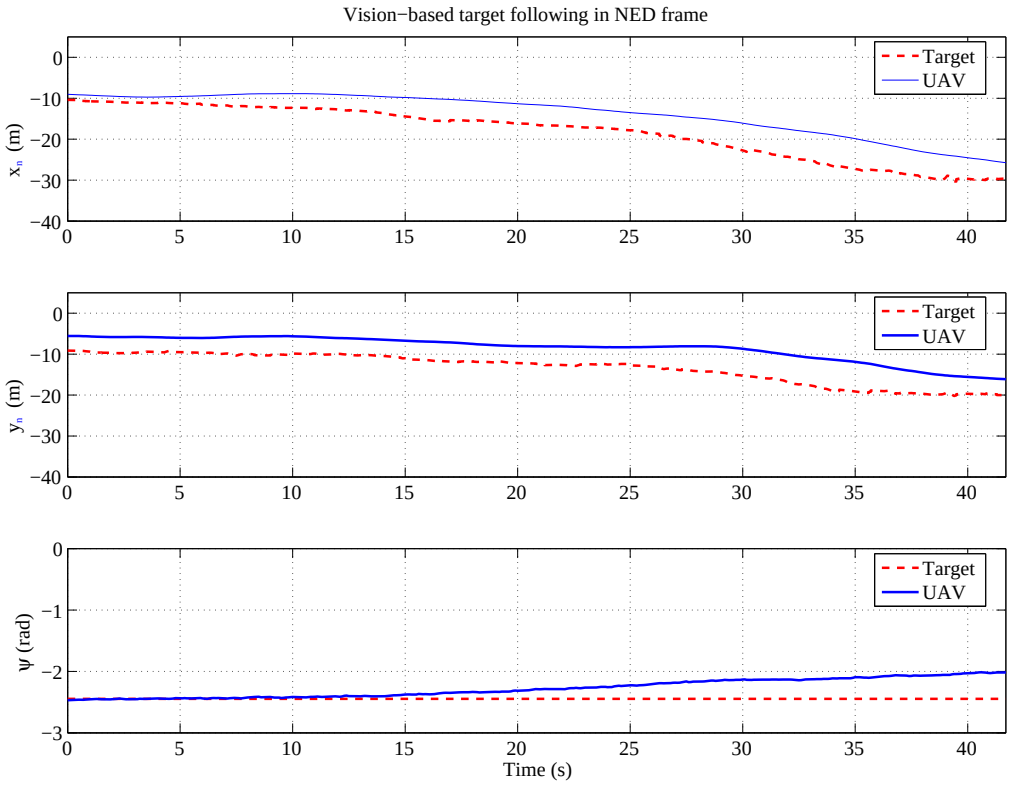


Figure 7.15: Test result of vision-based target tracking.

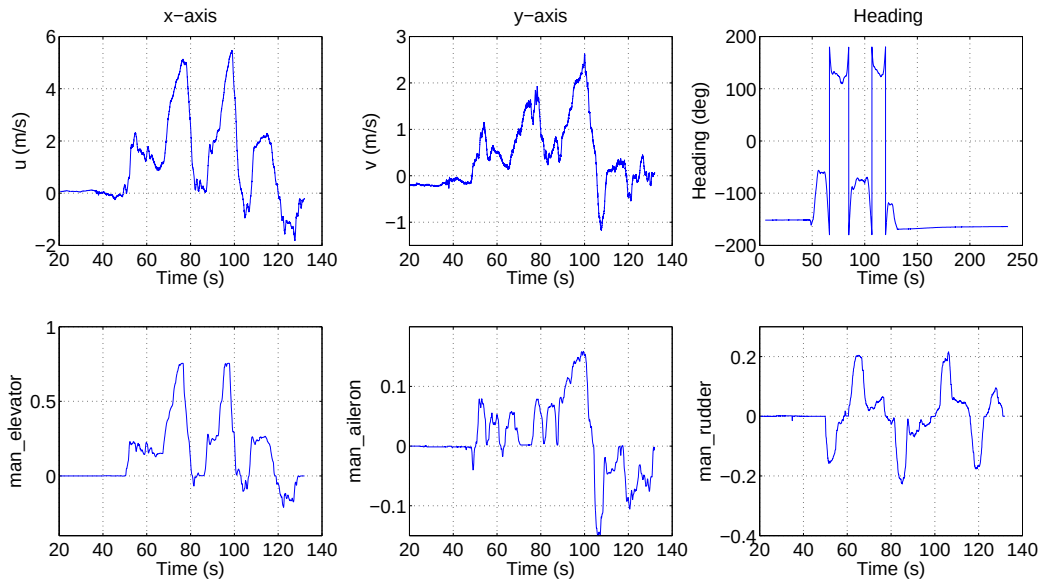


Figure 7.16: GremLion semi-auto flight performance

velocity u and y-axis velocity v are in accordance with the changing trend of the manual elevator and aileron, respectively.

Next the switch between semi-auto and full-auto should be conducted to examine the software implemented control logic. The flight scheduling is as listed below: 1. Send command to set GremLion to semi-auto mode; then the pilot performs the manual flight; 2. After the manual flight testing, a mode switch command is sent to UAV to switch from semi-auto mode to full-auto mode to start performing a height climb behavior up to 5 m; 3. Once the GremLion finishes the auto climb-up task, GCS operator then issues the mode switch command again to set back to semi-auto mode and pilot gently descends the UAV.

The control behaviors during the flight are all listed in Fig. 7.17. The three listed dots represents the behaviors activated by GCS operator. The first behavior is semi-auto mode at time 74.1 s, then the full-auto mode behavior is triggered at 108.7 s, and the ground user switches back to semi-auto mode at 126.1 s.

Fig. 7.18 demonstrates the flight performance with mode switch. It is clear that at time

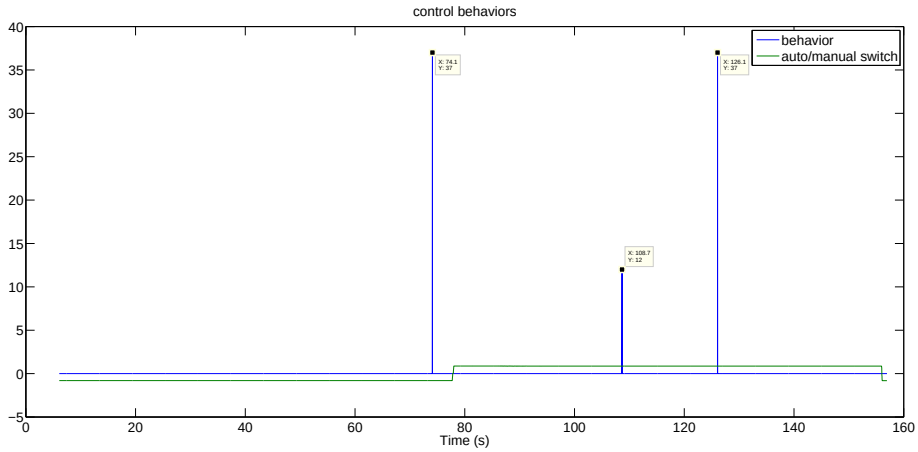


Figure 7.17: GremLion flight with mode switch - behavior

of mode transition from semi-auto to full-auto, the position reference and velocity reference are both generated from the software module *Auto reference generation* listed in Fig. 3.14. The dashed rectangle area highlights the references generated given the current position and desired position, where the height is a ramp signal while both x-axis and y-axis keep the current position. When the mode is switched to semi-auto mode, the manual signal generated velocity reference needs to start calculating given the current velocity, which is highlighted in the ellipse area in the x-axis graph. In both cases, the controllers for inner-loop and outer-loop remain unchanged while only the references for outer-loop are modified. Besides, the mode switch is performed in the near hover condition. As such, there is no transition period during switching and the stability and safety of mode switch can be guaranteed. In conclusion, all the detailed specifications to realize the complicated mode-switch control are perfectly accomplished by the flexible software systems.

7.6 Conclusion

In this chapter, the developed software systems are throughly examined in various applications. The behavior-based control mechanism is verified via the full envelop flight including

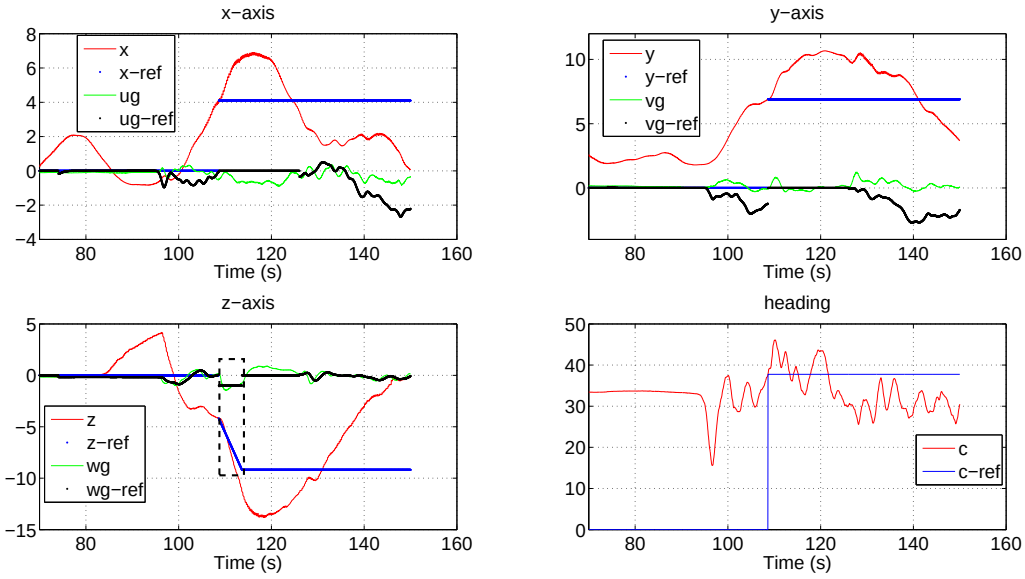


Figure 7.18: GremLion flight response with mode switch

engine related behaviors, take-off, path tracking and landing. A multiple-UAV formation application is also formulated and modeled with corresponding software diagrams. A vision-based tracking control of UAV is also implemented. Finally, a practical competition-oriented UAV flight control with mode switch is implemented. All the applications verify the successful design of the whole real-time software systems.

Chapter 8

Conclusions and Future Work

This research aims to conduct a comprehensive study on the real-time software system for unmanned vehicles. Specifically, a universal framework consisting of multiple UAVs and a ground control system is proposed. Based on the framework, the detailed specifications of UAV system are tackled and decomposed from a top-down perspective. Data flow diagrams are applied to decompose the specifications into subtasks, and then model the subtasks in an object-oriented way with UML diagrams. With the software modeling accomplished, the real-time onboard system can be implemented. In addition, the fundamental RTOS image customization and hardware layout design is also covered. Meanwhile, a user-friendly GCS system is developed to realize interactions between UAVs and GCS operator. To achieve formation flight, a hybrid communications network is also established. Finally, to verify the overall software system, various flight tasks including single UAV flight, and formation flight are conducted.

8.1 Contributions

The research work on comprehensive real-time software system development contributes in the following five aspects.

First, a universal framework for multiple unmanned vehicles is proposed. Within this

framework, various research works can be conducted. From the modeling and control of single unmanned vehicle, to the coordination and control of multiple vehicles. The framework consists of onboard systems and ground control system. Each system is further detailed with necessary modules within it. The onboard system is decomposed into the following modules: *Unmanned vehicle* represent the hardware platform, *Sensing* represents the onboard sensor components, *Flight control system* to depict the onboard control algorithms, *Simulation model* is used to replace true hardware vehicle in simulation mode, *Servo driving* is to output control signals to actuators, *Communications* is to communicate with GCS or other UAVs. For ground control system, *Information monitoring* is to display the flight data in various ways, *Task management* can realize high level mission planning. In all, this framework is complete, flexible and universal.

Next, we have designed the software platform including hardware layout and RTOS image customization. We have successfully designed a two-processor based avionics system to tackle all the challenging tasks including flight control, vision processing, formation flight. Specifically, a control subsystem platform and a vision subsystem platform are developed. The control subsystem is based on the industry standard QNX RTOS, while the vision subsystem is based on Linux OS. Both OSes are customized via sophisticated procedures. The customized OSes with rich hardware interfaces provide us more confidence to incorporate more advanced sensors in the near future.

Given the specifications of a real-time onboard avionics system, a top-down decomposition method is carried out with data flow diagrams. After decomposition, the subtasks of the onboard system is identified. Each task is modeled as an object based on UML diagrams. The task scheduler mechanism is designed to coordinate the execution order and time duration of each task. For flight control law implementation, a flexible behavior-based mechanism is designed. The structure consists of three layers, scheduling, outer-loop control law and inner-loop control law. Given different behaviors, the scheduling will generate the corresponding reference for outer-loop. For a mission plan consisting of multiple behaviors,

the scheduling layer will trigger each behavior accordingly until the plan is accomplished. A ground control system based on MFC is also developed which can realize various view including Google Maps and onboard camera live images. Interested researcher could follow our software design concepts to develop their own software system for a UAV/UGV.

We have also developed a hybrid communication network for cooperative tasks. The centralized network among GCS and UAVs can realize long range and point to point communication. The P2P network among UAVs can realize high bandwidth cooperative data exchange. We have designed a data format protocol to ensure efficient and reliable communication between sender and receiver. The data format can be used to represent data, user commands and live images. The sending and receiving functions are also modeled with data flow diagrams. Interested reader could use the data format protocol and data flow diagrams to establish their own communication network.

Finally, a leader-follower based formation flight is implemented with the software systems. A carefully designed communication protocol for formation is proposed. The message sequence diagram in UML is applied to describe the message passing protocols between leader and follower. A cooperative formation module is implemented with the specially designed protocol and can be deployed either on the leader or the follower. The communication protocol can also be applied in other cooperative tasks with small modifications.

8.2 Future Work

Although a comprehensive study on real-time software systems have been conducted, it is only the staring point of our UAV research. Considering more challenging tasks, it should be reasonable to extend the work to the following directions.

Multi-Core Support

The current control subsystem and vision subsystem are implemented on a single core processor. With multi-core processor available on the market, it is worthy to off-load current

onboard program to be executed on different cores to increase processing speed. Actually, the OMAP3530 processor comes with a DSP core, which is especially suitable for performing image related processings.

Indoor Localization and Navigation

Indoor implementation of small-scale UAV helicopter is of great interest and potential. But due to the lack of localization signal indoor, the progress is still in the initial stage. Currently our NUS UAV research team is undergoing this challenge. A micro-aerial-vehicle (MAV), FeiLion is constructed with laser scanner and onboard camera to realize localization and navigation. Due to large amount of data retrieved from laser scanner and camera, the computation burden becomes an issue for current processor. To solve this, we are exploring more advanced processors to port current applications to new processors.

Formation Control with Path Planning

The current approach to formation flight control is under the assumption that the follower trajectory will not collide with the leader. However, this issue is of great importance in practical flight tests. As such, an online path planning method which can realize collision avoidance is necessary such that the formation flight is more robust and reliable.

Bibliography

[1] S. Bayraktar, G. E. Fainekos, and G. J. Pappas, "Hybrid Modeling and Experimental Cooperative Control of Multiple Unmanned Aerial Vehicles," *43rd IEEE Conference on Decision and Control*, Atlantis, Bahamas, Vol. 4, pp. 4292-4298, 2004.

[2] T. X. Brown, S. Doshi, S. Jadhav, Jesse Himmelstein, "Test Bed for a Wireless Network on Small UAVs," *Proceedings of AIAA 3rd Unmanned Unlimited Technical Conference*, pp. 20-23, September 2004.

[3] W. Burgard, M. Moors, D. Fox, R. Simmons, and S. Thrum, "Collaborative multi-robot exploration," *Proceedings IEEE International Conference Robotics and Automation*, pp. 476-481, 2000.

[4] G. Cai, B. M. Chen and T. H. Lee, *Unmanned Rotorcraft Systems*, Springer, New York, 2011.

[5] G. Cai, B. M. Chen, K. Peng, M. Dong and T. H. Lee, "Modeling and control system design for a UAV helicopter," *Proceedings of the 14th Mediterranean Conference on Control and Automation*, vol. 2, pp. 600-606, Ancona, Italy, 2006.

[6] G. Cai, B. M. Chen, and T. H. Lee, "Design and implementation of robust flight control system for a small-scale UAV helicopter," *7th Asian Control Conference*, pp. 691-697, Hong Kong, 2009.

[7] G. Cai, B. M. Chen, T. H. Lee, and K. Y. Lum, "Comprehensive Nonlinear Modeling of an Unmanned-Aerial-Vehicle Helicopter," *AIAA Guidance, Navigation and Control Conference and Exhibit*, 18 - 21 August 2008, Honolulu, Hawaii.

[8] G. Cai, K. Peng, B. M. Chen and T. H. Lee, "Design and assembling of a UAV helicopter system," *Proceedings of the Fifth International Conference on Control and Automation*, vol. 2, pp. 697-702, Budapest, Hungary, 2005.

[9] B. M. Chen, *Robust and H_∞ Control*, London, UK: Springer-Verlag, 2000.

[10] B. M. Chen, Z. Lin and K. Liu, "Robust and perfect tracking of discrete-time systems," *Automatica*, Vol. 38, No. 2, pp. 293-299, February 2002.

[11] D. T. Cole, A. H. Goktogan and S. Sukkarieh, "The demonstration of a cooperative control architecture for UAV teams," *Experimental Robotics*, Springer Tracts in Advanced Robotics Volume 39, pp. 501-510, 2008.

[12] J. E. Cooling, *Real-time Software Systems - An Introduction to Structured and Object-oriented Design*, PWS Publishing Company, 1997.

[13] M. Dong, B. M. Chen, G. Cai, K. Peng, "Development of a Real-time Onboard and Ground Station Software System for a UAV Helicopter," *Journal of Aerospace Computing, Information and Communication*, Vol. 4, pp. 933-955, 2007.

[14] M. Dong, B. M. Chen, C. Cheng, "Development of 3D monitoring for an unmanned aerial vehicle," *Proceedings of 1st International Conference on Computer Science and Education*, pp.135-140, Xiamen, China, 2006.

[15] M. Dong, Z. Sun, "A behavior-based architecture for unmanned aerial vehicles," *Proceedings of International Symposium on Intelligent Control*, pp. 149-155, Taipei, Taiwan, 2004.

[16] X. Dong, B. M. Chen, G. Cai, H. Lin and T. H. Lee, "A comprehensive real-time software system for flight coordination and control of multiple unmanned aerial vehicles," *International Journal of Robotics and Automation*, Vol. 26, pp. 49-64, February 2011.

[17] B. P. Douglass, *Real-Time UML: Developing Efficient Objects for Embedded Systems*, Addison-Wesley, MA, 1999.

[18] D. B. Edwards, T. A. Bean, D. L. Odell and M. J. Anderson, "A leader-Follower Algorithm for Multiple UAV Formations," *Proceedings of 2004 IEEE/OES Autonomous Underwater Vehicles*, Sebasco Estates, Maine, pp. 517-523, 2004.

[19] A. H. Fagg, M. A. Lewis, J. F. Montgomery, and G. A. Bekey, "The USC Autonomous Flying Vehicle - an Experiment in Real-time Behavior-based Control," *Proceedings of the 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Yokohama, Japan, pp. 1173-1180, 1993.

[20] J. Feddema and D. Schoenwald, "Decentralized control of cooperative robotic vehicles," Presented at the SPIE, Vol. 4364, Aerosense, Orlando, FL, 2001.

[21] H. Gomaa, "A software design method for real-time systems," *Communications of the ACM*, Vol. 27, pp. 938-949, 1984.

[22] G. Hoffmann, D. G. Rajnarayan, S. L. Waslander, et al., "The Stanford testbed of autonomous rotorcraft for multi agent control (STARMAC)," *23rd Digital Avionics Systems Conference*, Vol. 2, pp. 12.E.4.1-12.E.4.10, 2004.

[23] J. P. How, "Multi-vehicle flight experiments: recent results and future directions," *Symposium on Platform Innovations and System Integration for Unmanned Air, Land and Sea Vehicles*, AVT-146, Florence, Italy, 2007.

[24] J. Heinlein, "Optimized Multiprocessor Communication and Synchronization Using a Programmable Protocol Engine," Technical Report No. CSL-TR-98-759, Stanford University, March 1998.

[25] J. P. How, E. King, and Y. Kuwata, "Flight Demonstrations of Cooperative Control for UAV Teams," *Proceedings of the 3rd AIAA Unmanned Unlimited Technical Conference, Workshop and Exhibit*, Chicago, IL, AIAA-2004-6490, 2004.

[26] IEEE 802.11 Working Group, "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications," *IEEE 802.11 standard, including 802.11a and 802.11b extensions*, September 1999.

[27] J. S. Jennings, G. Whelan, and W. F. Evans, "Cooperative search and rescue with a team of mobile robots," *Proceedings of IEEE International Conference Advanced Robotics*, pp. 193-200, 1997.

[28] C. J. John, "Automatic Formation Flight Control System (AFFCS) – A System For Automatic Formation Flight Control of Vehicles Not Limited to Aircraft, Helicopters, or Space Platforms," *U.S. Patent 6,926,233*, Sept. 08, 2005.

[29] J. S. Jang, and C. J. Tomlin, "Design and Implementation of a Low Cost Hierarchical and Modular Avionics Architecture for the DragonFly UAVs," *Proceedings of the AIAA Guidance, Navigation, and Control Conference and Exhibit*, AIAA-2002-4465.

[30] R. Krten, *Getting Started with QNX Neutrino 2 - A Guide for Real-time Programmers*, PARSE Software Devices, Kanata, Ontario, 1999.

[31] D. J. Kruglinski, *Inside Visual C++ (4th edition)*, Microsoft Press, Redmond, Washington, 1995.

[32] F. Lin, X. Dong, B. M. Chen, K. Y. Lum and T. H. Lee, "A robust real-time embedded vision system on an unmanned rotorcraft for ground target following," *IEEE Transactions on Industrial Electronics*, Vol. 59, No. 2, pp. 1038-1049, February 2012.

[33] A. D. Mali, "On the Behavior-based Architectures of Autonomous Agency," *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 32, No. 3, pp. 231-242, 2002.

[34] T. W. McLain and R. W. Beard, "Unmanned air vehicle testbed for cooperative control experiments," *Proceedings of American Control Conference*, Boston, MA, Vol. 6, pp. 5327-5331, 2004.

[35] K. Peng, M. Dong, B. M. Chen, G. Cai, K. Y. Lum and T. H. Lee, "Design and implementation of a fully autonomous flight control system for a UAV helicopter," *Proceedings of the 26th Chinese Control Conference*, Zhangjiajie, Hunan, China, Vol. 6, pp. 662-667, 2007.

[36] *OMAP35x Applications Processor Technical Reference Manual*, Texas Instruments, 2008.

[37] *QNX Neutrino RTOS v6.3, System Architecture*, Sixth Edition, QNX Software Systems Corporation.

[38] A. Ryan, X. Xiao, S. Rathinam, et al., "A Modular Software Infrastructure for Distributed Control of Collaborating UAVs," *Proceedings of the AIAA Guidance, Navigation, and Control Conference and Exhibit*, Keystone, Colo, Vol. 5, pp. 3248-3256, 2006.

[39] D. Rus, B. Donald, and J. Jennings, "Moving furniture with teams of autonomous robots," *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 235-242, 2000.

[40] M. Scheutz, and V. Andronache, "Architectural Mechanisms for Dynamic Changes of Behavior Selection Strategies in Behavior-based Systems," *IEEE Transactions on Systems, Man and Cybernetics - Part B: Cybernetics*, Vol. 34, No. 6, pp. 2377-2395, 2004.

[41] P. Seiler, A. Pant and K. Hedrick, "Analysis of bird formations," *Proceedings of the 41st IEEE Conference on Decision and Control*, Las Vegas, Nevada USA, December 2002.

[42] T. Shima and S. Rasmussen, "UAV Cooperative Decision and Control, Challenges and Practical Approaches", SIAM, Philadelphia, 2009.

[43] D. Shreiner, M. Woo, J. Neider and T. Davis, *OpenGL Programming Guide*, Addison-Wesley, Boston, 1997.

[44] *Spread Spectrum Wireless Data Transceiver User Manual*, Version 6.3, Revision 1, FreeWave Technologies, Inc.

[45] B. Wang, X. Dong, B. M. Chen, T. H. Lee and S. K. Phang, "Formation Flight of unmanned rotorcraft based on robust and perfect tracking approach," *Proceedings of the 2012 American Control Conference*, Montreal, Canada, pp. 3284-3290, June 2012.

[46] B. Wang, X. Dong and B. M. Chen, "Cascaded control of 3D path following for an unmanned helicopter," *Proceedings of the 2010 IEEE International Conference on Cybernetics & Intelligent Systems*, Singapore, pp. 70-75, June 2010.

[47] ArduPlane, *http://code.google.com/p/ardupilot-mega/*.

[48] ArduPilot mission planner, *http://code.google.com/p/ardupilot-mega/wiki/Mission*.

[49] Bitbake, *http://en.wikipedia.org/wiki/BitBake*.

[50] CentOS, the community enterprise operating system, *http://www.centos.org/*.

[51] Gumstix developer center, *http://www.gumstix.org*.

[52] Gumstix tutorial, *http://www.gumstix.org/getting-started.html*.

[53] GremLion, *http://uav.ece.nus.edu.sg/gremlion.html*.

[54] OpenEmbedded, *http://www.openembedded.org/wiki/Main_Page*.

[55] OpenPilot, *http://www.openpilot.org/*.

[56] Paparazzi, the free autopilot, *http://paparazzi.enac.fr/wiki/Main_Page*.

[57] PC104 Consortium, *http://www.pc104.org/specifications.php.*

[58] Pixhawk, *https://pixhawk.ethz.ch/.*

[59] Pontech UAV100, *http://www.pontech.com/details/138.*

[60] Predator, *http://science.howstuffworks.com/predator6.htm.*

[61] QGroundControl station, *http://qgroundcontrol.org/.*

[62] QNX Board Supporting Package, *http://community.qnx.com/sf/wiki/do/viewPage/projects.bsp/wiki/BSPAndDrivers/.*

[63] QNX Neutrio RTOS, *http://www.qnx.com/.*

[64] RTLinux, *http://www.rtlinuxfree.com/.*

[65] Telit company, *http://www.telit.com/en/.*

[66] UAVForge, *http://www.uavforge.net/uavhtml/.*

[67] VxWorks RTOS, *http://www.windriver.com/products/vxworks/.*

[68] WPA supplicant, *http://hostap.epitest.fi/wpa_supplicant/.*

[69] Wikipedia, peer-to-peer, *http://en.wikipedia.org/wiki/Peer-to-peer.*

[70] Wikipedia, RQ-4 Global Hawk, *http://en.wikipedia.org/wiki/Northrop_Grumman_RQ-4_Global_Hawk.*

[71] Wikipedia, UAV, *http://en.wikipedia.org/wiki/Unmanned_aerial_vehicle.*

Published Papers

Refereed Journal Articles:

1. X. Dong, B. M. Chen, G. Cai, H. Lin and T. H. Lee, "A comprehensive real-time software system for flight coordination and control of multiple unmanned aerial vehicles," *International Journal of Robotics and Automation*, Vol. 26, pp. 49-64, February 2011.

2. F. Lin, X. Dong, B. M. Chen, K. Y. Lum, and T. H. Lee, "A robust real-time embedded vision system on an unmanned rotorcraft for ground target following," *IEEE Transactions on Industrial Electronics*, Vol. 59, No. 2, pp. 1038-1049, February 2012.

3. G. Cai, B. M. Chen, X. Dong and T. H. Lee, "Design and implementation of a robust and nonlinear flight control system for an unmanned helicopter," *Mechatronics*, Vol. 21, No. 5, pp. 803-820, August 2011.

4. G. Shirazi, P. Wang, X. Dong and C. Tham, "Target Tracking with QoS Support in Large Wireless Sensor Networks," *Wireless Sensor Network*, Vol. 1, No. 5, pp. 370-382, 2009.

International Conference Articles:

1. X. Dong, M. Dong, B. Wang, B. M. Chen and T. H. Lee, "A comprehensive software architecture for unmanned aerial vehicles," *Proceedings of the 2011 IEEE*

International Conference on Service Operations, Logistics, and Informatics, Beijing, China, pp. 595-600, July 2011.

2. X. Dong, G. Cai, F. Lin, B. M. Chen, H. Lin and T. H. Lee, "Implementation of formation flight of multiple unmanned aerial vehicles," *Proceedings of the 8th IEEE International Conference on Control and Automation*, Xiamen, China, pp. 904-909, June 2010.

3. X. Dong, B. M. Chen, G. Cai, H. Lin and T. H. Lee, "Development of a comprehensive software system for implementing cooperative control of multiple unmanned aerial vehicles," *Proceedings of 7th IEEE International Conference on Control and Automation*, Christchurch, New Zealand, pp. 1629-1634, December 2009.

4. B. Wang, X. Dong, B. M. Chen, T. H. Lee and S. K. Phang, "Formation flight of unmanned rotorcraft based on robust and perfect tracking approach," *Proceedings of the 2012 American Control Conference*, Montreal, Canada, pp. 3284-3290, June 2012.

5. A. Karimoddini, X. Dong, G. Cai, F. Lin, H. Lin, B. M. Chen and T. H. Lee, "A composed hybrid structure for the autonomous flight control of unmanned helicopters," *Proceedings of the 18th IFAC World Congress*, Milan, Italy, pp. 2632-2637, August-September 2011.

6. B. Wang, X. Dong and B. M. Chen, "Cascaded control of 3D path following for an unmanned helicopter," *Proceedings of the 2010 IEEE International Conference on Cybernetics & Intelligent Systems*, Singapore, pp. 70-75, June 2010.